IRIS-4D System Administrator's Guide

IRIS-4D Series

SiliconGraphics
Computer Systems

Document number: 007-0603-010

IRIS-4D System Administrator's Guide

Version 1.0

Document Number 007-0603-010

Technical Publications:

Marcia Allen
Kathleen Chaix

Special Thanks to the Technical Marketing Group

IRIS-4D System Administrator's Guide
Version 1.0
Document Number 007-0603-010

Silicon Graphics, Inc.
Mountain View, California

Contents

Introduction

This guide describes procedures used in the administration of an AT&T 3B2 Computer running the UNIX System V Release 3.0 operating system. It is designed to accomplish the following objectives:

- provide clear instructions on how to perform the administrative tasks of a UNIX system
- give you background information about when and why these tasks are desirable
- serve as a quick reference to administrative procedures

What Is a System Administrator?

A System Administrator performs two main tasks:

- decides what rules are needed to govern the use of the computer system
- implements those rules so as to provide the maximum amount of computing service for the system's users, consistent with the physical limitations of the machine

Obviously, if you are the only user of your 3B2 Computer, these tasks consist simply of those things you do to keep the machine running and your programs and data from disappearing permanently. If, on the other hand, your 3B2 Computer will be used by a number of other people, the tasks become more complex, and you are required to be aware of the needs of your whole user community.

Some Assumptions About Your Experience

This guide assumes that you know the mechanics of using a computer terminal to enter commands, and that you have an awareness of such UNIX system fundamentals as the directory structure and the shell. We also expect that you feel comfortable using the 3B2 Computer; you know how to turn it on and how to use such things as the diskette drive. (If you feel unsure of your knowledge on these points, you might find it helpful to refer to the *Documentation Roadmap*. The *Documentation Roadmap* contains titles and descriptions of other UNIX system manuals you may want to look at before starting on this one.

How This Guide is Organized

There are two main parts to this guide:

Part 1 - Procedures	Part 1 contains ten sets of step-by-step procedures that tell you how to keep your 3B2 Computer in operation. Each set of tasks is related to a general topic, such as User Services or Processor Operations.
Part 2 - Support Information	Part 2 contains ten chapters of more detailed information about each of the ten sets of procedures. The chapters are numbered to parallel the procedures of Part 1.

In the back of the book there are three appendices, a glossary, and an index.

How to Use This Guide

1. **Use the procedures in Part 1 to begin with.**

   They lead you through administrative tasks without requiring preliminary knowledge or experience in that area.

2. **Use the chapters in Part 2 to learn more about what the procedures do.**

   They explain what is going on in the procedures and provide background information about the basic elements of the UNIX system.

3. **Use the Index for navigation.**

   It provides a cross-reference so you can uncover all the places in the guide where a topic is discussed.

4. **Use the Glossary to look up definitions of terms that are unfamiliar.**

5. **As you gain experience, use the guide for reference.**

About the Procedures

A table at the beginning of each procedure gives you information in capsule form. Table entries appear only when the information is relevant. The tables follow this style:

Purpose	Summary of what the procedure is used for.
When Performed	When you should schedule the procedure.
Starting Conditions	The state the computer should be in when you begin the procedure. Any special login requirements.
sysadm menu	The part of the System Administration Menu package that contains the subcommands to perform the procedure.
Commands	The commands used to perform the procedure.
Firmware Programs	The names of programs to be run in firmware mode.
Bootable Programs	The names of programs used to boot the system.
Media	Diskettes or tapes used in the procedure.
Time	Approximately how long the procedure takes.
Caution	Special instructions you must follow before or during the procedure to ensure the integrity of your system software and user files.
Reference	The chapter and section in Part 2 where this topic is more fully discussed. Other UNIX system manuals where additional information can be found.

System Administration Commands

The majority of the procedures are based on menus of the System Administration Menu package. This package consists of a hierarchical arrangement of interactive screens that lead you through system administration tasks. It is described in the *Owner/Operator Manual.*

The procedures shown in this guide by-pass the higher level System Administration Menus and take you directly to the subcommands. Subcommands are the equivalent of menu selections from lower level menus. If you prefer, you may start at the main menu with the unadorned command

```
$ sysadm
```

You can get to the submenu level with a command like

```
$ sysadm filemgmt
```

System States

In some procedures, we state that a particular system state is required. In most cases this means that the system must be in either the single- or multi-user state. The single-user state corresponds to run level 1, while the multi-user state corresponds to run levels 2 or 3. Procedures for bringing the system to different system states are found in Part 1 under Processor Operations Procedures. See the section on "Operating Levels," in Chapter 3, Processor Operations, for more information on system states.

Logins

In some procedures, we state that a particular login is required. This frequently means that you must be logged in as **root** to do the procedure. The phrase "an authorized login" is also used. The standard meaning of this term is that you must log in using a special administrative or system login name to do the procedure (see Chapter 1, System Security, for a list of these logins).

Passwords

It is strongly recommended that you set up and use passwords for administrative and system logins (see Procedure 1.4 for information on how to do this). In the procedures, we assume that such password protection has been established. When you enter a **sysadm** command as an ordinary user, therefore, you are prompted for a password. We show this with an entry like:

```
$ sysadm backup
Password:
```

At this point, to go ahead with the procedure, you are required to enter an acceptable password. As is always the case in the UNIX system, the password is not echoed to your screen.

In procedures that require you to be logged in as **root** (that is, the super-user), you are not prompted for the **sysadm** password. Also, the pound sign (#) prompt is used for the **root** login. Here's an example:

```
# sysadm backup
```

Notation Conventions

Whenever the text includes examples of output from the computer and/or commands entered by you, we follow the standard notation scheme that is common throughout UNIX system documentation:

- Text that you type in from your terminal is shown in **bold** type.
- Text that is printed on your terminal by the computer is shown in constant width type.
- Comments and explanations within a display are shown in *italic* type and are indented to separate them from the text that represents computer output or input.

  Italics are also used to show substitutable values, such as *file*, when the format of a command is shown.
- There is an implied **RETURN** at the end of each command and menu response you enter.
- Where you may be expected to enter only a **RETURN** (as in the case where you are accepting a menu default), the symbol <CR> is used.

- The dollar sign ($) and pound sign (#) symbols are the standard default prompt signs for an ordinary user and root, respectively.

  $ – means you are logged in as an ordinary user.
  # – means you are logged in as root

- When the # prompt is used in an example, it means the command illustrated may be used only by root.
- When the full path name of a command is shown in an example (like /etc/fsck) the command must be entered that way.

Command References

When commands are mentioned in a section of the text for the first time, a reference to the manual section where the command is described is included in parentheses: **command**(section). The numbered sections are located in the following manuals:

Sections (1), (6)	*IRIS-4D User's Reference Manual*
Sections (1), (1M), (7), (8)	*IRIS-4D System Administrator's Reference Manual*
Sections (1), (2), (3), (4), (5)	*IRIS-4D Programmer's Reference Manual*

Information in the Examples

While every effort has been made to present displays of information just as they appear on your terminal, it is possible that your system will produce slightly different output. Some displays depend on a particular machine configuration that may differ from yours. Changes between releases of the UNIX system software may cause small differences in what appears on your terminal.

Maintaining System Security

This chapter deals with maintaining the security of your computer system. It includes:

Security Guidelines	Security guidelines for setting passwords and permissions to protect the system from unauthorized access.
Logins and Passwords	Aging passwords to Control the amount of time passwords may be kept for logins, locking logins to prevent unauthorized use, and protecting administrative commands and logins with passwords.
Set-UID and Set-GID	Preventing unauthorized use of programs conditioned to execute via administrative logins Set-UID and Set-GID.

Important Security Guidelines

The security of the system is ultimately the responsibility of all who have access to the system. Some security items to consider are:

- Anyone with physical access to the machine can literally walk off with it.
- Set the access permissions to directories and files to allow only the necessary permissions for owner, group, and others.
- Give all logins passwords and change passwords regularly. Do not pick obvious passwords. Six-to-eight character nonsense strings using letters and numbers are recommended over standard names. Logins that are not needed should be either removed or blocked.
- Systems with dial-up ports should have logins but even then a system with dial-up ports is not totally secure. Sensitive information should not be kept on a system with dial-up ports.
- Users who frequently use the **su** command can compromise the security of the system by accessing files belonging to other users without consent. This command is also dangerous since you must know another user's login password to use it. The more users who know a given login and password, the less secure access is to the system. For this reason, a log is kept on the use of the command. Check the file /usr/adm/sulog to monitor use of the **su** command. The format of /usr/adm/sulog is described in Appendix A, Directories and Files.
- Login directories, **.profile** files, and files in **/bin**, **/usr/bin**, and **/etc** that are writable by others are not secure.
- Encrypt sensitive data files. The **crypt**(1) command together with the encryption capabilities of the editors (**ed** and **vi**) provide protection for sensitive information.
- Log off the system if you must be away from the data terminal. Do not leave a logged-in console or terminal unattended, especially if you are logged in as root.

Logins and Passwords

The discussion of logins and passwords covers the following:

- Password aging
- Sample **/etc/passwd** entries
- Locking unused logins
- Special administrative logins

Password Aging

The password aging mechanism forces users to change their password on a periodic basis. It will also prevent a user from changing a new password before a specified time interval. Password aging is selectively applied to logins by editing the **/etc/passwd** file. Realistically, password aging forces a user to adopt at least two passwords for a login. If you require more access control than what is provided by password aging, you can change **/etc/profile** to require a second access code as part of the login process.

The password aging information is appended to the encrypted password field in the **/etc/passwd** file. The password aging information consists of a comma and up to four bytes (characters) in the format:

,Mmww

The meaning of these fields is as follows:

,	The delimiter between the password itself and the aging information.
M	The Maximum duration of the password
m	The minimum time interval before the existing password can be changed by the user (in weeks, following the notation in Figure 1-1).

ww The week (counted from the beginning of 1970) when the password was last changed and two characters, *ww*, are used. You do not enter this information. The system automatically adds these characters to the password aging information.

All times are specified in weeks (0 through 63) by a 64-character alphabet. Figure 1-1 shows the relationship between the numerical values and character codes. Any of the character codes may be used in the four fields of the password aging information.

Character	Number of Weeks
. (period)	0 (zero)
/ (slash)	1
0 through 9	2 through 11
A through Z	12 through 37
a through z	38 through 63

Figure 1-1: Password Aging Character Codes

Two special cases apply for the character codes:

- If *M* and *m* are equal to zero (the code, ..), the user is forced to change the password at the next login. No further password aging is then applied to that login.
- If *m* is greater than *M* (for example, the code, ./), only **root** is able to change the password for that login.

Sample /etc/passwd Entries

Password administration can be set up in a variety of ways to meet the needs of different organizations. Some examples are discussed in the following sections.

Changing Passwords Every Two Weeks

The following shows the password aging information required to establish a new password every 2 weeks (0) and to deny changing the new password for 1 week (/).

1. Here is a typical login/password entry in the **/etc/passwd** file for the typical user **jqu**:

   jqu:RTKESmMOE2m.E:100:1:J. Q. Username:/usr/jqu:

2. To cause **jqu** to change the password at least every 2 weeks, but keep it at least for 1 week, you should use the code **0/**. After you edit the **/etc/passwd** file, adding **,0/** to the password field, the entry looks like this:

   jqu:RTKESmMOE2m.E,0/:100:1:J. Q. Username:/usr/jqu:

   After the password entry is changed, **jqu** will have to change the password at the next login and every 2 weeks thereafter.

3. After **jqu**'s first login following the change, the system automatically adds the two-character, "last-time-changed" information to the password field.

   jqu:RTKESmMOE2m.E,0/W9:100:1:J. Q. Username:/usr/jqu:

   In this example, **jqu** changed the password in week **W9**.

Changing the Password One Time Only

The following shows the password aging information required to establish for one time only a new password when the user next logs in (using the .. code).

1. Here is the typical login/password entry for user **jqu** again:

   jqu:RTKESmMOE2m.E:100:1:J. Q. Username:/usr/jqu:

2. To cause **jqu** to change the password at the next login (and to cause this only once), you should use the code ... After you edit the **/etc/passwd** file, adding **,..** to the password field, the entry looks like this:

   jqu:RTKESmMOE2m.E,..:100:1:J. Q. Username:/usr/jqu:

   After the password entry is changed, **jqu** will have to change the password at the next login only.

3. After **jqu** supplies the new password, the system automatically removes the aging code (,..) from the password field:

   jqu:EFDNLqsFUjfs:100:1:J. Q. Username:/usr/jqu:

   Note that the encrypted password information has changed.

Changing the Password with root Login

The following shows the password aging information required to establish a password for a login so that only the root login can change the password (using the ./ code).

1. Here is the typical login/password entry for user **jqu** again:

   jqu:RTKESmMOE2m.E:100:1:J. Q. Username:/usr/jqu:

2. To prevent **jqu** from changing the password, you should use the code **./**. After you edit the **/etc/passwd** file, adding **,./** to the password field, the entry looks like this:

   jqu:RTKESmMOE2m.E,./:100:1:J. Q. Username:/usr/jqu:

   Now only **root** can change the password for the **jqu** login. If **jqu** tries to change the password, a "permission denied" message is displayed.

Locking Unused Logins

If a login is not used or needed, you should remove the entry from **/etc/passws** or disabling (locking) the login.

A login is locked by editing the **/etc/passwd** file and changing the encrypted password field to contain one or more characters that are not used by the encryption process. One way to do this is to use the expression **Locked;**. In this expression, the semicolon (;) is an unused encryption character. The text, however, only serves to remind you that the login is locked. Another expression, **not valid**, could also be used to prevent the use of a login because a space is another unused encryption character.

The following line entry from the **/etc/passwd** file shows the "locked" **bin** login.

*bin:*Locked;*:2:2:0000-Admin(0000):/bin:*

Special Administrative Passwords

There are two familiar ways to access the system: either via a conventional user login or the **root** login. If these were the only two ways to access the system, however, effective use of the system would have to be curtailed (because **root** would own many directories) or many users would have to know the **root** password (a bad security risk) or the system would be wide open (because **root** would own few directories). All of these conditions are undesirable.

The solution to a good mix of system use and system security is available through the use of special system logins and administrative commands that can be password-protected. The commands, which are also logins, that can be password-protected perform functions that might be needed by a number of users on your system:

Function	Use
setup	This command is used to set up the computer. Once the machine has been set up, you do not want anyone doing it again without your knowledge.
sysadm	This command allows access to many useful administrative functions that do not require a user to log in as root.
powerdown	This command powers the computer down.

The commands above allow access to selected directories and system functions. They may be used as login names as well as commands. If you log in to the system with one of these names, the system will execute the command after login and exit to the **login** prompt once you quit or complete the function performed by the command.

Most of these system functions allow a user access to critical portions of the operating system. For this reason, it is recommended that you assign passwords to the commands above. Once you assign passwords to these commands, any user attempting to log on using one of these commands as a login (and any user attempting to execute one of these commands from the shell) is prompted for the password.

It is recommended that the passwords to these commands/logins be known to only a few users.

Login	Use
root	This login has no restrictions on it and it overrides all other logins, protections, and permissions. It allows the user access to the entire operating system. The password for the root login should be very carefully protected.
sys	This login has the power of a normal user login over the files it owns, which are in /usr/src.
bin	This login has the power of a normal user login over the files it owns, which are in /bin.
adm	This login has the power of a normal user login over the object files it owns, which are in /usr/adm.
uucp	This login owns the object and spooled data files in /usr/lib/uucp.
nuucp	This login is used by remote machines to log into the system and initiate file transfers via /usr/lib/uucp/uucico.
rje	This login has an entry in /etc/passwd. There are no files currently associated with this login.
daemon	This is the login of the system daemon, which controls background processing.
trouble	This login has the power of a normal user login over the files it owns, which are in /usr/lib/trouble.
lp	This login owns the object and spooled data files in /usr/spool/lp.

Set-UID and Set-GID

The set-user identification (set-UID) and set-group identification (set-GID) bits must be used very carefully. These bits are set through the **chmod**(1) command and can be specified for any executable file. When a user runs an executable file that has either of these bits set, the system gives the user the permissions of the owner of the executable.

System security can be compromised if a user copies another program onto a file with **-rwsrwxrwx** permissions. For example, if the switch user (su) command has the write access permission allowed for others, anyone can copy the shell onto it and get a password-free version of su. The following paragraphs provide a few examples of command lines that can be used to identify the files with a set-UID.

For more information about the set-UID and set-GID bits, see **chmod**(1) and **chmod**(2).

Check Set-UIDs Owned by root

The following command line lists all set-UID programs owned by **root**. The results are mailed to **root**. All mounted paths are checked by this command starting at /. Any surprises in root's mail should be investigated.

```
# find / -user root -perm -4100 -exec ls -l {} \;| mail root
you have mail
# mail
From root Mon Aug 27 07:20 EDT 1984
-r-sr-xr-x   1 root     bin        38836 Aug 10 16:16 /usr/bin/at
-r-sr-xr-x   1 root     bin        19812 Aug 10 16:16 /usr/bin/crontab
-r-sr-xr-x   1 root     bin        27748 Aug 10 16:16 /usr/bin/shl
---s--x--x   1 root     sys        46040 Aug 10 15:18 /usr/bin/ct
-r-sr-xr-x   1 root     sys        12092 Aug 11 01:29 /usr/lib/mv_dir
-r-sr-sr-x   1 root     bin        33208 Aug 10 15:55 /usr/lib/lpadmin
-r-sr-sr-x   1 root     bin        38696 Aug 10 15:55 /usr/lib/lpsched
---s--x---   1 root     rar        45376 Aug 18 15:11 /usr/rar/bin/sh
-r-sr-xr-x   1 root     sys        11416 Aug 11 01:26 /bin/mkdir
-r-sr-xr-x   1 root     sys        11804 Aug 11 01:26 /bin/rmdir
-r-sr-xr-x   1 root     bin        12524 Aug 11 01:27 /bin/df
-rwsr-xr-x   1 root     sys        21780 Aug 11 01:27 /bin/newgrp
-r-sr-sr-x   1 root     sys        23000 Aug 11 01:27 /bin/passwd
-r-sr-xr-x   1 root     sys        23824 Aug 11 01:27 /bin/su
? d
#
```

In this example, an unauthorized user (**rar**) has made a personal copy of **/bin/sh** and has made it set-UID to **root**. This means that **rar** can execute **/usr/rar/bin/sh** and become the super user.

Check Set-UIDs in the Root File System

The following command line reports all files with a set-UID for the root file system. The **ncheck**(1M) command, by itself, can be used on a mounted or unmounted file system. The normal output of the **ncheck** –s command includes special files. Here, the **grep** command is used to remove device files from the output. The filtering done in this example to remove the device files is applicable only for the root file system. The output of the modified **ncheck** is used as an argument to the ls command. The use of the ls command is possible only if the file system is mounted.

```
# ls -l 'ncheck -s /dev/root| cut -f2| grep -v dev'
-r-sr-xr-x   1 root     bin        12524 Aug 11 01:27 /bin/df
-rwxr-sr-x   1 root     sys        32272 Aug 10 15:53 /bin/ipcs
-r-xr-sr-x   2 bin      mail       32852 Aug 11 01:28 /bin/mail
-r-sr-xr-x   1 root     sys        11416 Aug 11 01:26 /bin/mkdir
-rwsr-xr-x   1 root     sys        21780 Aug 11 01:27 /bin/newgrp
-r-sr-sr-x   1 root     sys        23000 Aug 11 01:27 /bin/passwd
-r-xr-sr-x   1 bin      sys        27964 Aug 11 01:28 /bin/ps
-r-xr-sr-x   2 bin      mail       32852 Aug 11 01:28 /bin/rmail
-r-sr-xr-x   1 root     sys        11804 Aug 11 01:26 /bin/rmdir
-r-sr-xr-x   1 root     sys        23824 Aug 11 01:27 /bin/su
-r-xr-sr-x   1 bin      sys        21212 Aug 10 16:08 /etc/whodo
#
```

In this example, nothing looks suspicious.

Check Set-UIDs in Other File Systems

The following command line entry shows the use of the **ncheck** command to examine the **usr** file system (**/dev/root**, assuming a single-disk system with default partitioning) for files with a set-UID. In this example, the complete pathnames for the files start with **/usr**. **/usr** is not part of the **ncheck** output.

```
# ncheck -s /dev/root | cut -f2
/dev/dsk/c1d0s2:
/bin/at
/bin/crontab
/bin/shl
/bin/sadp
/bin/timex
/bin/cancel
/bin/disable
/bin/enable
/bin/lp
/bin/lpstat
/bin/ct
/bin/cu
/bin/uucp
/bin/uuname
/bin/uustat
/bin/uux
/lib/mv_dir
/lib/expreserve
/lib/exrecover
/lib/accept
/lib/lpadmin
/lib/lpmove
/lib/lpsched
/lib/lpshut
/lib/reject
/lib/sa/sadc
/lib/uucp/uucico
/lib/uucp/uusched
/lib/uucp/uuxqt
/rar/bin/sh
#
```

In this example, the **/usr/rar/bin/sh** should be investigated.

Providing User Services

This chapter deals with providing a variety of services to the users of your UNIX system, including:

Login Administration	Assigning user and group IDs to persons authorized to be users of your system and maintaining the **/etc/passwd** and **/etc/group** files.
The Environment	Setting up a master profile and helping users develop individual profiles and establish environment variables.
Communication Services	Establishing and maintaining such communications services as message-of-the-day, news, and mail.
Anticipating Requests	Developing an organized plan for responding to user problems.

Login Administration

Adding Users

Before users are permitted to log in to your system, they must be listed in the **/etc/passwd** file. The **adduser** selection from the **sysadm usermgmt**(1) menu leads you through a series of prompts that create an entry in the **/etc/passwd** file. If you prefer, you can make the necessary changes to the **passwd** file yourself using an editor. You need to login as **root** to do this; **/etc/passwd** is generally installed as a read-only file. An entry in the **passwd** file consists of a single line with seven colon-separated fields.

```
abc:gOQ3xsv05bWuM,9/TA:103:123:Allen B. Cipher:/usr/abc:
```

The fields are:

login name	A valid name for logging onto the system. A login name is from three to six characters; the first character must be alphabetic. It is usually chosen by the user.
password	The encrypted form of the password, if any, associated with the login name in the first field. All encrypted passwords occupy 13 bytes. The actual password can be a maximum of 8 characters. At least one character must be numeric. This is to discourage users from choosing ordinary words as passwords. When you add a user to the file you may use a default password, such as **passwd9**, and instruct the user to change it at the first login. Following the encrypted password, separated by a comma, there may be a field that controls password aging. See Password Aging in Chapter 1, System Security.
user id	The user-ID number (**uid**) is between 0 and 50,000. The number must not include a comma. Numbers below 100 are reserved. User-ID 0 is reserved for the super-user. The System Administration menu package does not permit you to specify a number below 100 when adding a user.

group id	The same conditions apply to the group-ID (gid) number as to the **uid**, except that the group-ID 1 is reserved for the "other" group.
account	The account is the name of and optional additional information about the user. There is no required format for this field.
home directory	The home directory is where the user is placed upon logging in. The name is usually the same as the login name, preceded by a parent directory such as **/usr**. The **modadduser** menu of the System Administration package allows you to specify the default parent directory. The home directory is the origination point of the user's directory tree.
program	This is the name of a program invoked at the time the user logs in. If the field is empty, the default program is **/bin/sh**. This field is most commonly used to invoke a special shell, such as /bin/csh (C shell).

As noted above, the password field may contain a subfield that controls the aging of passwords. A description of how the process works can be found in Chapter 1 and in **passwd**(4) in the *IRIS-4D Programmer's Reference Manual.* The effect is to force users periodically to select a new password. If password aging is not implemented, a user can keep the same password indefinitely.

If you inspect the **/etc/passwd** file on your workstation you will see several commands listed among the user login names. These are commands, such as **sysadm**(1), that can have passwords assigned to them.

Changing or Deleting Password Entries

As with adding users, there are **sysadm usermgmt** menu selections for changing or deleting user entries from the **/etc/passwd** file. You have the option, however, of using an editor to make the changes.

Every so often a user will forget his or her password. When that happens (let's say to user **abc**), you can login as root and enter the command:

```
# passwd abc                    (The # prompt shows you are root.)
New password: passwd9           (The password entered is not echoed.)
Re-enter new password: passwd9
```

Since you did this as the super-user (**root**), you were not prompted for the old password. The command changes **abc**'s password to **passwd9**. You should make sure the user changes the password immediately.

When you delete a login from **/etc/passwd** using the sysadm **usermgmt** menu, all of the user's files and directories are removed. If you remove an entry from the **passwd** file using an editor, you have only removed the entry. The user's files remain.

Group IDs

Group IDs are a means of establishing another level of ownership of and access to files and directories. Users with some community of interest can be identified as members of the same group. Any file created by a member of the group carries the group-ID as a secondary identification. By manipulating the permissions field of the file, the owner (or someone with the effective user-ID of the owner) can grant read, write, or execute privileges to other group members.

Information about groups is kept in the **/etc/group** file. A sample entry from this file is shown and explained below:

```
prog::123:jqp,abc
```

Each entry is one line; each line has the following fields:

group name	The group name can be from 3 to 8 characters, the first of which must be alphabetic.
password	The password field should not be used.
group id	The group id is a number from 0 to 50,000. The number must not include a comma. Numbers below 100 are reserved.
login names	The login names of group members are in a comma-separated list. A login name should be a member of no more than one group. There is nothing to prevent a user from having more than one login name, however, as long as each is unique within the system.

The User's Environment

For Bourne shell users, the key element in establishing an environment in which users can successfully communicate with the computer is the profile. This section describes Bourne shell profiles only.

For C shell users, two files, **.login** and **.cshrc**, replace the profile file. For more information on the C shell files, see An Introduction to the C Shell in the *IRIS-4D User's Guide*.

Bourne profiles are of two types:

1. The system profile.

   This is an ASCII text file, **/etc/profile**, that contains commands, shell procedures, and environment variables. Whenever a user logs in, the **login** process executes this file.

2. An individual user's profile.

   This is an executable commands file, **.profile**, that may reside in a user's home directory. The individual profile can contain additional commands and variables that further customize a user's environment. If one exists, it too is executed at login time, after the execution of **/etc/profile. in this chapter.**

   A sample **/etc/profile** is shown in Figure 2-1.

```
# The profile that all logins get before using their own .profile.
trap "" 2 3
export LOGNAME
# Login and -su shells get /etc/profile services.
# -rsh is given its environment in its .profile.
case "$0" in
-su)
                        export PATH
                        ;;
-sh)
                        export PATH
                        # Allow user to break the Message of the Day only.
                        trap "trap '' 2" 2
                        cat -s /etc/motd
                        trap "" 2
                        if mail -e
                        then
                        echo "you have mail"
                        fi
                        if [ ${LOGNAME} != root ]
                        then
                        news -n
                        fi
                        ;;
esac
umask 022
trap 2 3
```

Figure 2-1: A Default **/etc/profile**

Several interesting things are contained in the profile:

- Some environment variables are exported (see Environment Variables later in this chapter).
- A file named **/etc/motd** is **cat**-ted (see Message of the Day later in this chapter).
- If the user is **not root**, the names of news items are displayed (**news -n**; see **news** later in this chapter).

- If the user has mail (**mail -e**), a message about it is displayed (see Mail later in this chapter).

For information on the shell programming commands used in Figure 2-1, see sh(1) in the *IRIS-4D User's Reference Manual.*

Environment Variables

An array of strings called the environment is made available by exec(2) when a process begins. Since **login** is a process, the array of environment strings is made available to it. An example of a typical array of strings for the Bourne shell is shown in Figure 2-2.

```
PS1=$
LOGNAME=abc
PWD=/usr/abc
HOME=/usr/abc
PATH=:/bin:/usr/bin:
SHELL=/bin/sh
MAIL=/usr/mail/abc
TERM=5420
PAGER=pg
TZ=EST5EDT
TERMINFO=/usr/lib/terminfo
EDITOR=vi
```

Figure 2-2: Environment Array for a Typical User

The environment variables shown in Figure 2-2 give values to 12 names for user **abc**. Other programs make use of the information. For example, the user's terminal is defined as a 5420 (TERM=5420). When the user invokes the editor **vi**(1), **vi** checks the file referenced by TERMINFO (**/usr/lib/terminfo**) where it learns the characteristics of a 5420 terminal (such as the 24-line screen). New strings can be defined at any time. By convention they are defined with the variable in uppercase, followed by an equal sign, followed by the value. Once defined, an environment variable can be made global for the user through the **export** statement. The individual **.profile** file can contain whatever the user wants.

umask

A system default controls the permissions mode of any files or directories created by a user. The workstation has default values of 666 for files and 777 for directories. That means that for files everyone automatically gets read and write permission. For directories, everyone gets read, write, and execute permission. (Execute permission on a directory means the ability to **cd** to the directory and to copy files from it.)

Users frequently set up a user mask in their startup file by means of the **umask**(1) command. **umask** alters the default permission levels by a specified amount. For example,

```
umask 027
```

leaves the permission level for owner unchanged, lowers the permission level for group by 2, and reduces the permissions for others to zero. The system default was 666; this user mask changes it to 640, which translates into read and write permission for the owner, read permission for the group, and no permission for others.

For Bourne shell users, there may be a **umask** command in **/etc/profile**. If there is, it does not change a user's ability to put one in **.profile**.

Default Shell and Restricted Shell

Generally, when a user logs in the default program that is started is **/bin/csh**. There may be cases, however, where a user needs to be given a restricted shell.

A restricted shell is one where the user is not allowed to

- change directories
- change the value of $PATH
- specify pathnames or command names containing a slash (/). That is, the user of a restricted shell may not access files or directories other than the present working directory or those included in $PATH
- redirect output

The restrictions are enforced after **.profile** has been executed.

The administrator can use a restricted shell strategy to limit certain users to the execution of a small number of commands or programs. By setting up a special directory for executables (**/usr/rbin**, for example), and controlling PATH so it only references that directory, the administrator can restrict the user's activity in whatever way is appropriate.

User Communications Services

Several ways of communicating with and among users are available in thc UNIX system. Some of the most frequently used are described in this section.

Message of the Day

Items of broad interest that you want to make available to all users can be put in the **/etc/motd** file. The contents of **/etc/motd** are displayed on the user's terminal as part of the login process. The login process executes a file called **/etc/profile** (for the Bourne shell), which is an executable shell script that, among other things, commonly contains the command

cat /etc/motd

Any text contained in **/etc/motd** is displayed for each user each time the uscr logs in. For this information to have any impact on users, you must take pains to use it sparingly and to clean out outdated announcements. A typical use for the Message of the Day facility might be

```
5/30:  The system will be unavailable from 6-11pm Thursday, 5/30
- preventive maintenance.
```

Part of the preventive maintenance should be to remove the notice from **/etc/motd**.

news

Another electronic bulletin board facility is the **/usr/news** directory and the **news**(1) command. The directory is used to store announcements in text files, the names of which are usually used to provide a clue to the content of the news item. The **news** command is used to print the items on your terminal.

The **/etc/profile** file is also used to inform users about news itcms. A typical **/etc/profile** contains the line

news –n

The **–n** argument causcs the names of files in the **/usr/news** directory to be printed on a user's terminal as the user logs in. Item names are displayed only for current items, that is, items added to the **/usr/news** directory since the user last looked at the news. The idea of currency is implemented like this: when you read a news item an empty file named **.news_time** is written in your login directory. As with any other file, **.news_time** carries a time stamp indicating the date and time the file was created. When you log in, a comparison is made between the time stamp of

your **.news_time** file and time stamp of items in **/usr/news**.

Unlike the Message of the Day where users have no ability to turn the message off, with **news** users have a choice of several possible actions:

read everything	If the user enters the command, **news** with no arguments, all news items posted since the last time the user typed in the command are printed on the user's terminal.
select some items	If the **news** command is entered with the names of one or more items as arguments, only those items selected are printed.
read and delete	After the **news** command has been entered, the user can stop any item from printing by pressing the delete key. Pressing the delete key twice in a row stops the program.
ignore everything	If the user is too busy to read announcements at the moment, they can safely be ignored. Items remain in **/usr/news** until removed. The item names will continue to be displayed each time the user logs in.
flush all items	If the user simply wants to eliminate the display of item names without looking at the items, a couple of techniques will work: **$ touch .news_time** updates the time-accessed and time-modified fields of the **$ news > /dev/null** prints the news items on the null device.

write to All Users

The ability to write to all logged-in users, via the wall(1M) command, is an extension of the **write**(1) command. It is fully effective only when used by the super-user. While **wall** is a useful device for getting urgent information out quickly, users tend to find it annoying to have messages print out on their terminal right in the middle of whatever else is going on. The effect is not destructive, but is somewhat irritating. Many users guard against this distraction by including the command

mesg n

in their **.profile**. This blocks other ordinary users from interjecting a message into your **stdout**. The **wall** command, when used by the super-user, overrides the **mesg n** command. It is best to reserve this for those times when you as the system administrator need to ask users to get off the system.

mail

The UNIX system offers an electronic mail utility through which users can communicate among themselves. If your system is connected to others by networking facilities, **mail**(1) can be used to communicate with persons on other systems.

Anticipating User Requests

As the system administrator for your workstation you can expect users to look to you to help solve any number of problems. In addition to the system log described in Chapter 3, Processor Operations, you will find it helpful to keep a user trouble log. The problems that users run into fall into patterns. If you keep a record of how problems were resolved, you will not have to start from scratch when a problem recurs.

Trouble Reporting

Another technique that is strongly recommended is an organized way for users to report problems. Figure 2-3 shows a sample Trouble Report that can be used to record and keep track of system problems.

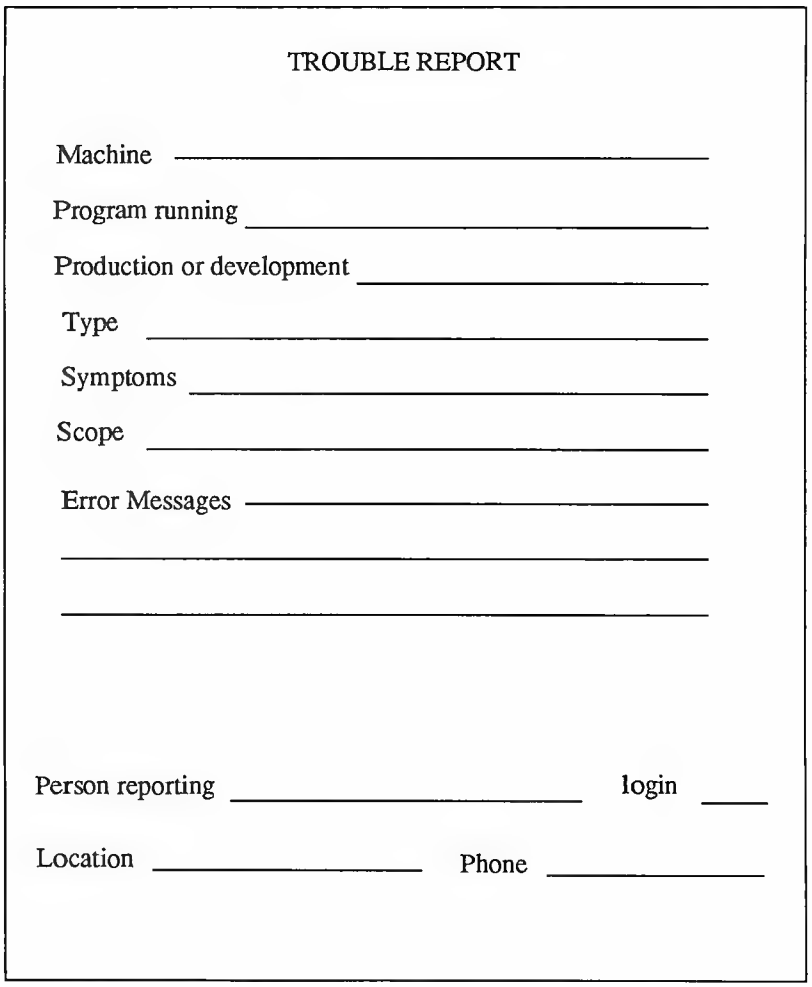


Figure 2-3: A Sample Trouble Report

Day-to-Day Operations

This chapter deals with the day-to-day operations of your workstation.

- General operating policy

  Guidelines for balancing the needs of system maintenance and the interests of your user community; suggestions for record keeping; lists of important administrative directories and files.

- Operating Levels

  Definition of the operating levels of the system; how they are controlled.

General Operating Policy

Many administrative tasks require the system to be shut down to a run level other than the multi-user state (see the discussion on Operating Levels below). This means that conventional users cannot access the system. When the machine is taken out of the multi-user state, the users on the machine at the time are requested to log off. You should do these types of tasks when they will interfere the least with the activities of the user community.

Sometimes situations arise that require the system to be taken down with little or no notice provided to the users. Try to provide the user community as much notice as possible about events affecting the use of the machine. When the system must be taken out of service, also tell the users when to expect the system to be available. Use the news (**/etc/news/***headline*) and the Message of the Day (**/etc/motd**) to keep users informed about changes in hardware, software, policies, and procedures.

At your discretion, the following items should be done as prerequisites for any task that requires the system to leave the multi-user state.

1. When possible, schedule service-affecting tasks to be done during periods of low system use. For scheduled actions, use the Message of the Day (**/etc/motd**) to inform users of future actions.
2. Check to see who is logged in before taking any actions that would affect a logged-in user. The **/etc/whodo** and **/bin/who** commands can be used to see who is on the system.
3. If the system is in use, provide the users advanced warning about changes in system states or pending maintenance actions. For immediate actions, use the **/etc/wall** command to send a broadcast message announcing that the system will be taken down at a given time. Give the users a reasonable

amount of time to terminate their activities and log off before taking the system down.

Maintaining a System Log

In a multi-user environment it is strongly recommended that a complete set of records be maintained. A system log book can be a valuable tool when trouble shooting transient problems or when trying to establish system operating characteristics over a period of time. Some of the things that you should consider entering into the log book are:

- maintenance records (dates and actions)
- printouts of error messages and diagnostic phases
- equipment and system configuration changes (dates and actions)

The format of the system log and the types of items noted in the log should follow a logical structure. Think of the log as a diary that you update on a periodic basis. To a large measure, how you use your system will dictate the form and importance of maintaining a system log.

Administrative Directories and Files

This section briefly describes the directories and files that are frequently used by a system administrator. For more detail about the purpose and contents of these directories and files, see Appendix A, Directories and Files. For additional information on the formats of the system files, refer to Section 4 of the UNIX System V manual pages in the *IRIS-4D Programmer's Reference Manual*.

Root Directories

The directories of the **root** file system (/) are as follows.

bin	Directory that contains public commands.
boot	Directory that contains configurable object files created by the **/etc/mkboot**(1M) program.
dev	Directory containing special files that define all of the devices on the system.

etc	Directory that contains administrative programs and tables.
lib	Directory that contains public libraries.
lost+found	Directory used by **fsck**(1M) to save disconnected files.
tmp	Directory used for temporary files.
usr	Directory used to mount the **/usr** file system.

Important System Files

The following files and directories are important in the administration of the your system.

/etc/fstab	File used to specify the file system(s) to be mounted by **/etc/mountall** and remote file systems to be mounted by **/etc/rmountall**.
/etc/gettydefs	File containing information used by **/etc/getty** to set the speed and terminal settings for a line.
/etc/group	File describing each group to the system.
/etc/init.d	Directory containing executable files used in upward and downward transitions to all system run levels. These files are linked to files beginning with **S** (start) or **K** (stop) in **/etc/rc*n*.d**, where *n* is replaced by the appropriate run level.
/etc/inittab	File containing the instructions to define the processes created or terminated by **/etc/init** for each initialization state.
/etc/master.d	Directory containing files that define the configuration of hardware devices, software drivers, system parameters, and aliases.
/etc/motd	File containing a brief Message of the Day, output by **/etc/profile**.
/etc/passwd	File identifying each user to the system.
/etc/profile	File containing the standard (default) environment for all users.

/etc/rc0	File executed by **/etc/shutdown** that executes shell scripts in **/etc/rc0.d** and **/etc/shutdown.d** directories for transitions to system run-level 0.
/etc/rc0.d	Directory containing files executed by **/etc/rc0** for transitions to system run-level 0. Files in this directory are linked from files in the **/etc/init.d** directory and begin with either a **K** or an **S**. **K** indicates processes that are stopped, and **S** indicates processes that are started when entering run-level 0.
/etc/rc2	File executed by **/etc/init** that executes shell scripts in **/etc/rc2.d** and **/etc/rc.d** on transitions to system run-level 2.
/etc/rc2.d	Directory containing files executed by **/etc/rc2** for transitions to system run-levels 2 and 3. Files in this directory are linked from files in the **/etc/init.d** directory and begin with either a **K** or an **S**. **K** indicates processes that should be stopped and **S** indicates processes that should be started when entering run-levels 2 or 3.
/etc/rc.d	Directory containing executable files that do the various functions needed to initialize the system to run-level 2; they are executed when **/etc/rc2** is run. (Files contained in this directory prior to UNIX System V Release 3.0 were moved to **/etc/rc2.d**. This directory is only maintained for compatibility.)
/etc/shutdown	File containing a shell script that gracefully shuts down the system in preparation for system backup or for scheduled downtime.
/etc/TIMEZONE	File used to set the time zone shell variable TZ.
/etc/utmp	File containing the information on the current run-state of the system.
/etc/wtmp	File containing a history of system logins.
/usr/adm/sulog	File containing a history of **su** command usage. This file should be checked periodically for size.
/usr/lib/cron/log	File containing a history of all the actions taken by **/etc/cron**. This file should be checked periodically for size.

/usr/lib/spell/spellhist	File containing a history of all words that spell fails to match (if the Spell Utilities are installed on the system).
/usr/news	Directory containing news files. This directory should be checked periodically, and old files should be discarded.
	Directory containing crontab files for the **adm**, **root**, and **sys** logins and ordinary users listed in **cron.allow**.

Each of these files is described in more detail in Appendix A.

Operating Levels

General

After you have set up your worstation for the first time (plugging it in, hooking all the hardware together, booting it, running the setup programs,) as documented in the *IRIS-4D Series Owner's Guide*, you and other users can use the system. When you bring the system to multiuser mode, the following things happen: Whenever you turn it on (including the first time), the system comes up in a multi-user environment in which

- The file systems are mounted.
- The **cron** daemon is started for scheduled tasks.
- The basic networking functions of **uucp** are available for use.
- The spooling and scheduling functions of the LP package (if added to the system) are available for use.
- Users can log in. The **gettys** are spawned on all connected terminal lines listed in **/etc/inittab** to have **gettys** respawned. (**gettys** are not on when the system is installed. You have to turn them on yourself.)

This is defined as the multi-user state. It is also referred to as **init** state 2 because all of the activities of initializing the system are under the control of the **init** process. The "2" refers to entries in the special table **/etc/inittab** used by **init** to initialize the system to the multi-user state.

Not all activities, however, can be performed in the multi-user state. For example, if you were able to unmount a file system while users were accessing it, you would cause a lot of data to be lost. Hence, for unmounting and other system administration tasks, there is a need for another state, the single-user state.

The single-user state is an environment in which only the console has access to the system and the root file system alone is mounted. You are free to do tasks that affect the file systems and the system configuration because you are the only one on the system.

There are other system states (see Figure 3-1), but first a note of clarification. One of the more confusing things about the discussion of system states is that there are many terms used to identify the same thing: the particular operating level of the system.

Here is a list of frequently encountered synonyms:

- run state
  run level
  run mode
- init state
  system state

Likewise, each system state may be referred to in a number of ways, for example:

- single-user
- single-user mode
- run level 1, and so on

In any case, each state or run level clearly defines the operation of the computer. Figure 3-1 defines each of them.

Run Level	Description
0	Power-down state.
1, s, or S	Single-user mode is used to install/remove software utilities, run file system backups/restores, and to check file systems. Though **s** and **1** are both used to go to single user state, **s** only kills processes spawned by **init** and does not unmount file systems. State **1** unmounts everything except root and kills all user processes, except those that relate to the console.
2	Multi-user mode is the normal operating mode for the system. The default is that the **root** (/) and user (**/usr**) file systems are mounted in this mode. When the system is powered up it is put in multi-user mode.

Figure 3-1: System States

How init Controls the System State

UNIX systems always run in one state or another. The actions that cause the various states to exist are under the control of the **init** process, which is the first general process created by the system at boot time. It reads the file **/etc/inittab**, which defines exactly which processes exist for which run level.

In the case of the multi-user state (run level 2), **init** scans the file for entries that have a tag for the run level (the tag is a 2) and executes everything after the last colon (:) on the line containing the tag.

If you look at your **/etc/inittab**, you'll see something that looks like the following. (It is most unlikely that yours will look exactly like this one; **/etc/inittab** changes from one configuration to another.)

NOTE

If **/etc/inittab** was removed by mistake and is missing during shutdown, **init** will enter the single user state (**init s**). While entering single user state, **/usr** will remain mounted and processes not spawned by **init** will continue to run. You should replace **/etc/inittab** before changing states again.

The format of each line is

id:level:action:process

- *id* is one or two characters that uniquely identify an entry.
- *level* is zero or more numbers and letters (**0** through **6**, **s**, a, **b**, and **c**) that determines what *level*(s) *action* is to take place in. If *level* is null, the *action* is valid in all levels.
- *action* can be one of the following:

sysinit	run *process* before **init** sends anything to the system console (`Console Login:`).
bootwait	start *process* the first time **init** goes from single-user to multi-user state after the system is booted. (If **initdefault** is set to **2**, the process will run right after the boot.) *init* starts the process, waits for its termination and, when it dies, does not restart the process.
wait	when going to *level*, start *process* and wait until it's finished.

initdefault	when **init** starts, it will enter *level*; the *process* field for this *action* has no meaning.
once	run *process* once and don't start it again if it finishes.
powerfail	tells **init** to run *process* whenever a direct powerdown of the computer is requested.
respawn	if process does not exist, start it, wait for it to finish, and then start another.
ondemand	synonymous with **respawn**, but used only with *level* **a**, **b**, or **c**.
off	when in *level*, kill process or ignore it.

- *process* is any executable program, including shell procedures.
- # can be used to add a comment to the end of a line. Everything after a # on a line will be ignored by init.

When changing levels, **init** kills all processes not specified for that level. We'll go through more manageable pieces of this table below to get a clearer idea of how the system is controlled by **init**.

A Look at Entering the Multi-User State

Powering Up

When you power up your system, it enters single-user state by default. (You can change the default by modifying the **initdefault** line in your **inittab** file.) To start multi-user mode tye *multi*. In effect, going to the multi-user state follows these broad lines (see Figure 3-1):

1. The operating system is loaded and the early system initializations are started by **init**.
2. The run level change is prepared by the **/etc/rc2** procedure.
3. Finally the system is made public via the spawning of **getty**s along the terminal lines.

Early Initialization

Just after the operating system is first loaded into core via the specialized boot programs the **init** process is created. It immediately scans **/etc/inittab** for entries of the type **sysinit**:

```
zu::sysinit:/etc/bzapunix </dev/console >/dev/console 2>&1
fs::sysinit:/etc/bcheckrc </dev/console >/dev/console 2>&1
ck::sysinit:/etc/setclk </dev/console >/dev/console 2>&1
```

They are executed in sequence and perform the necessary early initializations of the system. Note that each entry indicates a standard input/output relationship with **/dev/console**. This is the way communication is established with the system console before the system has been brought to the multi-user state.

Preparing the Run Level Change

Now the system must be placed in a particular run level. First, **init** scans the table to find an entry that specifies an *action* of the type *initdefault*. If it finds one, it uses the run level of that entry as the tag it will use to select the next entries to be executed. In our sample **/etc/inittab**, the **initdefault** entry specifies run level 2 (the multi-user state) as the level to select and execute other entries:

```
is:2:initdefault:
s2:23:wait:/etc/rc2 >/dev/console 2>&1 </dev/console
co:234:respawn:/etc/getty console console
ct:234:off:/etc/getty contty 9600
he:234:respawn:sh -c 'sleep 20 ; exec /etc/hdelogger >/dev/console 2>&1'
l1:234:respawn:/etc/getty -t60 tty11 1200
l2:234:respawn:/etc/getty -t60 tty12 1200
l3:234:respawn:/etc/getty -t60 tty13 1200
```

The other entries shown above specify the actions necessary to prepare the system to change to the multi-user run level. First, **/etc/rc2** is executed. It executes all files in **/etc/rc2.d** that begin with the letter **S**. It then executes all the files in the **/etc/rc.d** directory, accomplishing (among other things) the following:

- sets up and mounts the file systems
- starts the **cron** daemon
- displays the current system hardware configuration
- makes **uucp** available for use
- makes line printer (lp) system available for use, if installed
- starts a **getty** for the Console
- starts the hard disk error logging daemon
- starts **getty** on the lines connected to the ports
- indicates (ttys)

At this moment, the full multi-user environment is established, and your system is available for users to log in.

A Look at the System Life Cycle

Changing Run Levels

In effect, changing run levels follows these broad lines:

1. The system administrator enters a command that directs **init** to execute entries in **/etc/inittab** for a new run level.
2. Key procedures, such as **/etc/shutdown**, **/etc/rc0**, and **/etc/rc2** are run to initialize the new state.
3. The new state is reached. If it is state 1 the system administrator can proceed.

Run Level Directories

Run levels 0 and 2 each have a directory of files that are executed in transitions to and from that level. These directories are **rc0.d**, **rc2.d**, and **rc3.d**, respectively. All files in these directories are linked to files in **/etc/init.d**. The run-level filenames look like this:

S*00name*

or

K*00name*

The filenames can be split into three parts:

S or K	The first letter defines whether the process should be started (S) or stopped (K) upon entering the new run level.
00	The next two characters are a number from 00 to 99. They indicate the order in which the files will be started (S00, S01, S02, etc.) or stopped (K00, K01, K02, etc).
name	The rest of the filename is the **/etc/init.d** filename this file is linked to.

For example, the **init.d** file **cron** is linked to the **rc2.d** file and **rc0.d** file **K70cron**. When you enter **init 2**, this file is executed with the **start** option: **sh S75cron start**. When you enter **init 0**, this file is executed with the **stop** option: **sh K70cron stop**. This particular shell script will execute **/usr/bin/cron** when run with the **start** option and **kill** the **cron** process when run with the **stop** option.

Because these files are shell scripts, you can read them to see what they do. You can modify the files, though it is preferable to add your own since the delivered scripts may change in future releases. To create your own scripts you should follow these rules:

- Place the file in **/etc/init.d**.
- Link the file to files in appropriate run state directories using the naming convention described above.
- Have the file accept the **start** and/or **stop** options.

Going to Single-User Mode

At times in a given work week, you will need to perform some administrative functions in the single-user mode, such as backing up the hard disk onto some diskettes (see Procedure 3.3). The normal way to go to single-user mode is through the **/etc/shutdown** command. This procedure executes all the files in **/etc/rc0.d** and **/etc/shutdown.d** directories by calling the **/etc/rc0** procedure, accomplishing, among other things, the following:

- closing all open files and stopping all user processes
- stopping all daemons and services
- writing all system buffers out to the disk
- unmounting all file systems except root

The entries for single-user processing in the sample **/etc/inittab** are:

```
s1:1:wait:/etc/shutdown -y -iS -g0 >/dev/console 2>&1 </dev/console
p1:s1234:powerfail:/etc/led -f   # start green LED flashing
p3:s1234:powerfail:/etc/shutdown -y -i0 -g0 >/dev/console 2>&1
```

There are two major ways to start the shutdown processing.

1. You can enter the **shutdown –i1** command (recommended).
2. You can enter the **init 1** command, which forces the **init** process to scan the table. The first entry it finds is the s1 entry, and it starts the shutdown processing.

Now the system is in the single-user environment, and you can perform the appropriate administrative tasks.

Turning the System Off

The final step in the life cycle of the system is turning it off. The following entries apply to powering the system down:

```
s0:056:wait:/etc/rc0 >/dev/console 2>&1 </dev/console
of:0:wait:/etc/uadmin 2 0 >/dev/console 2>&1 </dev/console
```

One way to turn the system off is to cause a powerfail signal to be sent to the system. You can either enter the **powerdown** command or directly invoke the **/etc/shutdown –i0** command.

In either case, the **/etc/shutdown** and **/etc/rc0** procedures are called to clean up and stop all user processes, daemons, and other services and to unmount the file systems. Finally, the **/etc/uadmin** procedure is called, which indicates that the last step—physically removing power from the system—is under firmware control.

Disk and Cartridge Tape Devices

This chapter covers what you need to know about the disk and cartridge tape devices on your workstation. The topics are:

- Disk device types and sizes
- Tape device sizes
- Making devices known to the operating system
- Formatting disks and tapes
- Verifying disk or tape usability

This chapter does not deal with file systems or the information stored on disk devices. Those subjects are covered in Chapter 5, File System Administration.

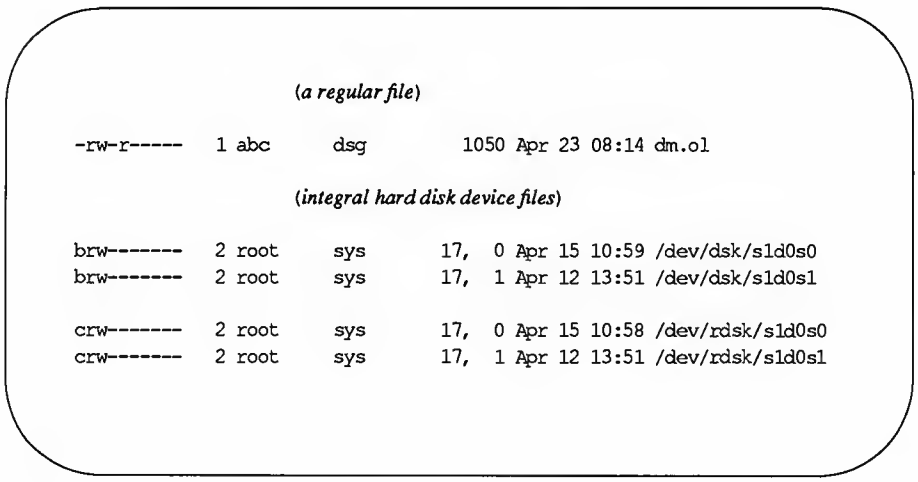
Device Types

All system software and user files are kept on the hard disk device(s). The cartridge tape device is used primarily for high-speed file system backups.

Identifying Devices to the Operating System

Before a disk or tape device can be used on a computer running the UNIX operating system, it must be made known to the system. For equipment that comes with your computer, the process of identifying devices is part of configuration and is done automatically as the system is booted.

The traditional way of handling the identification is through an entry in the /dev directory of the root file system. Of course, an entry in a directory is a file (or another directory), and conceptually a disk device is treated as if it were a file. There is a difference, however, which leads to the practice of referring to devices as "special" files. In the place where a regular file would show the character count for the file, for a special file you find two decimal numbers called the major and minor numbers. Figure 4-1 shows excerpts from the output of commands on a user's directory and the /dev directory structure.



(a regular file)

```
-rw-r-----   1 abc      dsg          1050 Apr 23 08:14 dm.ol
```

(integral hard disk device files)

```
brw-------   2 root     sys        17,  0 Apr 15 10:59 /dev/dsk/s1d0s0
brw-------   2 root     sys        17,  1 Apr 12 13:51 /dev/dsk/s1d0s1

crw-------   2 root     sys        17,  0 Apr 15 10:58 /dev/rdsk/s1d0s0
crw-------   2 root     sys        17,  1 Apr 12 13:51 /dev/rdsk/s1d0s1
```

Figure 4-1: Directory Listing Extracts: Regular and Device Files

The extracts from directory listings in Figure 4-1 show a regular file (indicated by the dash (-) in the first position of the line) with these characteristics:

- It has 1050 characters.
- The filename is **dm.ol**.

- It is owned by user **abc** who is a member of **dsg** group.
- The owner has read/write permission, group members have read permission, other users have no permissions.

There are also device files with these characteristics:

- Major and minor numbers appear in place of the character count.

  Major is the number of the device controller or driver (actually, an offset into a table of devices in the kernel); minor is the identifying number of the specific device.

- There are devices that have identical major and minor numbers, but they are designated in one entry as a block device (a `b` in the first column) and in another entry as a character device (a `c` in the first column).

  Notice that such pairs of files have different file names or are in different directories (for example, **/dev/dsk/s1d0s0** and **/dev/rdsk/s1d0s0**).

- There are alias names.
- The files are owned by **root**, and no group or other user has any permission to use them. This means that only processes with the **root** ID can read from and write to the device files. (The cartridge tape device is an exception to this rule.)

Block and Character Devices

The identification as a block device or a character device has more to do with how the device is accessed rather than with any physical characteristics. A block device name is used when the intent is to read from or write to the device in logical, 1024-byte blocks. In the UNIX system, standard C language subroutines for handling file I/O work with blocks.

A character device name is used to read from or write to the device one character at a time. A character device is also referred to as a "raw" device. This is reflected in the device names or directory names shown in Figure 4-1, where the character device version of the disk drive is in directory **/dev/rdsk**. The character-at-a-time method is used by some file maintenance utilities.

Defining a New Special File

The need to define new special device files occurs infrequently. If you add devices, the autoboot process takes care of defining the files. When the need does occur, however, there is a UNIX system command, **mknod**(1M), available to do it.

The general format of the **mknod** command is:

mknod *name* **b** | **c** *major minor*

mknod *name* **p**

The options of **mknod** are:

name	Specifies the *name* of the special file.
b	Specifies a block device.
c	Specifies a character device. The or sign (**|**) indicates you must specify one or the other.
major	The *major* number is the slot number.
minor	The *minor* number is the physical device.
p	Specifies the special file as a first-in, first-out device. This is also known as a named pipe. (For more on pipes, see the *IRIS-4D Programmer's Guide*.)

Formatting and Partitioning

Formatting a disk or tape means establishing addressable areas on the medium. Partitioning means assigning file systems or other logical units to addressable areas.

Formatting Disks and Tapes

Before a disk or tape can be used for the storage of information, it must be formatted. Until the medium is formatted, its surfaces are simply uncharted areas treated with a substance that accepts and holds magnetic charges. Formatting imposes an addressing scheme onto these magnetic surfaces. For disks, formatting maps both sides of the disk into tracks and sectors that can be addressed by the disk controller. A portion of the disk is reserved for data having to do with the specific disk. The volume table of contents resides in that area. The volume table of contents (VTOC) shows how the partitions on the disk are allocated. On a hard disk, an additional use of the reserved area is to map portions of the disk that may not be usable. Formatting a previously used disk, in addition to redefining the tracks, erases any data that may be there.

Hard Disk Partitioning

Hard disks are shipped from the factory already formatted. Partitions on the hard disk devices on your workstation are allocated in a standard arrangement. The arrangement varies according to whether you have one or two disk drives, and what size the drives are.

The first disk is partitioned to accommodate the **root** and **/usr** file systems, swap space.

The default partitions are fundamentally a compromise. After your system has been in operation for a few months, you may come to feel that a different arrangement would better serve the needs of your users.

Planning to Change Hard Disk Partitions

The basic question in reaching a decision about re-partitioning your hard disk devices is whether you would be better off having a larger number of smaller file systems, or staying with /usr and another user file system that takes up the entire second disk. Here are some subordinate questions that bear on the basic one:

- What group-IDs are defined? Do we have the right number of groups and are users assigned to them appropriately?
- What is the nature of the processing done by the user groups? Does their work require temporary data storage? Is there a big difference between the type of processing done by one group and that done by the others?
- Have we added, or are we planning to add, system software that changes our thinking about space requirements?

If you decide that re-partitioning is needed, it can be done with a full system restore. **fmthard** can be used to redefine partitions on disks other than the root disk.

Changing Partitions to Increase Swap Space

If you frequently get console messages warning of insufficient memory, it may mean that the system's current configuration of main memory and swap area is insufficient to support user demands. Before adding more main memory, an alternate solution is to expand the swap area (on either single or multiple hard disk systems).

This is accomplished by the full restore procedure described in Crash Recovery, in the *IRIS-4D Series Owner's Guide*. But before you run the procedure there are three things you should do first:

1. Find out about your present partitions.
2. Decide what the new partition sizes should be. (The disk is already fully allocated, so increasing the size of the swap partition means you have to reduce the size of one of the other partitions.)
3. Do a complete backup. (For information on backing up, refer to Backing Up and Restoring Files in the *IRIS-4D Series Owner's Guide*). (The process of changing partitions is going to erase everything that is presently there.)

You are now ready to go ahead with the back up procedure outlined in the *IRIS-4D Series Owner's Guide*.

The Bad Block Handling Feature

NOTE

This feature applies only to hard disk device(s). There is no comparable feature for diskettes or tapes.

The IRIS workstation has a software feature called bad block handling. The purpose of this feature is to extend the useful life of the integral hard disk by providing mechanisms for:

- detecting and remembering blocks that are no longer usable
- reminding you that you need to "fix" some remembered bad blocks
- restoring the usability of the disk in spite of the bad blocks that exist

It should be pointed out that new bad blocks seldom occur, particularly with the sealed environment of the integral disk, as long as you take reasonable precautions against movement or vibration of the computer while the disk is still spinning. But when a new bad block does occur, the data stored in the bad block is lost and the disk may be unusable in its current state.

The bad block handling feature addresses the problem of restoring the usability of the disk. However, you must address the data loss yourself with whatever backup procedures you deem appropriate for your situation. Backup procedures are also needed to protect against operational errors and other types of hardware failures. These other problems, particularly operational errors, are the dominant cause of data loss on a system. System backups provide protection against operational errors as well as protection against lost data due to new bad blocks. A discussion of backup procedures can be found in Chapter 5, File System Administration.

When Is a Block Bad?

A block is bad when it cannot reliably store data. This is discovered only when an attempt is made to read the data and the read fails. To make life more difficult, a read fail does not always guarantee a bad block. A read fail might also mean problems in the format of the disk or a failure in the controller or the hardware.

A write fail generally signals a problem with the format of the disk or a more basic failure in the disk or disk controller hardware. While all failures are reported, the bad block handling feature does not distinguish genuine bad blocks from format problems or hardware problems. To fix problems of those types you will need to reformat the disk or get the hardware repaired. In either case, you should call your service representative. Several distinct failures occurring about the same time

should also prompt you to contact your service representative to check them out.

What Makes a Block Unreliable?

A disk is an analog medium used to store digital data. The analog phenomena involve the magnetic properties of the film coating on disk surfaces. The data are recorded with a very high bit density to get millions of bits in a small space. Because of the density, the significance of small variations in the magnetic properties of the recording medium become magnified. The variations mean that a given portion of the medium may prefer to represent some bit patterns and dislike representing others. Normally, these preferences are insignificant compared to the signal level thresholds. When variations pass the point of being insignificant, the disk has a bad block. If the data pattern in a block matches the preferences in the recording medium, the bad block may escape recognition for a while. If the disk is active, however, the block will eventually be judged unreliable.

How Are Bad Blocks Fixed?

It is not really so much that a bad block is fixed, but that the system finds a way to live with it. A small portion of the disk is set aside from the normally accessible portion of the disk. This portion, call it the media-specific data area, cannot be reached by normal UNIX system commands and system calls. This reserved portion of the disk contains a description of the properties of the disk and other media-specific data.

The media-specific data portion of the disk includes a set of blocks called the surrogate image region. The mechanism for preserving the apparent accessibility of most disk blocks is to use surrogate image blocks to contain the data that were to have been stored in the bad blocks. The media-specific data also includes a mapping table that maps bad blocks on the disk to these surrogate image blocks. The disk driver software in the operating system translates disk accesses so the data are read from or written to the surrogate image block. This disk address translation is transparent to the calling software.

Most disks come with a few manufacturing defects. Bad blocks detected in the manufacturer's quality control checks are identified on a label when the unit is delivered. The bad block handling feature provides special software for remembering bad blocks that have been found and for mapping any additional ones that are found. If a surrogate block becomes bad, the software even remaps the original bad block to a new surrogate block.

A Few Blocks Cannot Be Mapped

A few special blocks, all in the media-specific data portion of the disk, cannot be mapped:

- the disk block containing the physical description of the disk
- the disk block(s) containing the mapping table

All other blocks, including surrogate image blocks, can be mapped.

When Are Bad Blocks Detected?

Bad blocks are detected when input/output disk operations fail for several successive attempts. This means that data being input or output are lost, but the system can restore use of the disk by mapping bad blocks to surrogate blocks that are readable.

Often Asked Questions

Why doesn't the system try to discover that a given block is bad while the system still has the data in memory?

Besides the undesirable increase in system size and complexity, severe performance degradation would result. Also, a block can become a bad block after the copy in memory no longer exists.

Why doesn't the system periodically test the disk for bad blocks?

Reading blocks with their current contents may not show a bad block to be bad. A thorough bit pattern test would take so long that you would never run it, even assuming a thorough test could be devised using ordinary write/read operations. The disk manufacturer already has tested the disk using extensive bit pattern tests and special hardware. All manufacturing defects have been dealt with already.

Why are disks with manufacturing defects used?

Allowing the disks to contain a modest number of manufacturing defects greatly increases the yield, thereby considerably reducing the cost. Many systems, including this one, take advantage of this cost reduction to provide a more powerful system at lower cost.

How Bad Block Handling Works

The bad block handling feature provides the mechanisms to detect, remember (where feasible), and add new bad blocks to the existing map. In normal operations the mechanisms are automated. Some cases do occur that require special handling, and even the automated cases have special properties at some stages of the processing. The remainder of this section describes:

- normal operation
- handling exceptional cases
- fixing bad blocks manually

Bad Block Handling: Normal Operation

The first phase of automated mechanisms involves detecting new bad blocks and remembering them for later mechanisms.

An understanding of the ways of referring to disk blocks helps in understanding the examples that follow.

- A block number is an integer counter.
- A physical block number is the integer counter that describes how the sectors are numbered on the disk. (For example, physical block 3 is sector 3 of head 0 of cylinder 0.)
- A logical block number is the counter for sectors, starting with 0 at the portion of the disk where disk partitions begin.
- A partition block number is the counter for sectors, starting with 0 at the beginning of the partition.
- A file system block number is the counter for file system blocks, starting with 0 at the beginning of the first partition in the file system.

A Bad Block Handling Scenario

In this scenario physical disk block 3 is a surrogate block for physical block X in the **/usr** file system.

Block X is in a text file. Block 3 has become a bad block sometime after the last time the file was read. Now the file needs to be read again. When block X is reached, the driver for the integral disk sees that block X is mapped to block 3 and attempts to read block 3. But block 3 is now bad and cannot be read. When the integral disk driver determines that block 3 is unreadable, the following messages are output to the system console.

```
WARNING: unreadable CRC hard disk error: maj/min = 17/0

        block # = 3

WARNING:
hard disk: cannot access sector 3, head 0, cylinder 0, on drive 0

Disk Error Daemon: successfully logged error for block 3 on disk
maj=17 min=0
```

CRC stands for Cyclic Redundancy Check, an error checking method.

The attempt to read the text file has failed. You notice the message on the console and run **shutdown**(1M) to go to single-user mode. While shutdown is running, the following message is output to the system console.

```
Disk Error Daemon: Disk maj=17 min=0: 1 errors logged
```

The following sections discuss what was happening in the scenario.

Disk Identification

These mechanisms support both single-disk and multi-disk models. To avoid confusion, and to support all possible configurations, disks are identified by their major/minor device numbers. Messages printed out by bad block handling use the major/minor numbers rather than any other name.

Detecting New Bad Blocks

The disk driver software detects the new bad blocks for the disk in question. The disk driver determines that a block is definitely not accessible by attempting several accesses. The disk driver also repositions the disk read/write heads between some of the retries to be sure that the problem is not a head positioning error. For information on reporting and logging new bad blocks, see Using the Bad Block Handling Feature in the *IRIS-4D Series Owner's Guide*.

File System Administration

How the File System is Organized

A primary function of the UNIX operating system is to support file systems. In the UNIX system, a file is a one-dimensional array of bytes with no other structure implied. Files are attached to a hierarchy of directories. A directory is merely another type of file that the user is permitted to use, but not to write; the operating system itself retains the responsibility for writing directories. The combination of directories and files make up a file system. Figure 5-1 shows the relationship between directories and files in a UNIX file system. The circles represent directories.

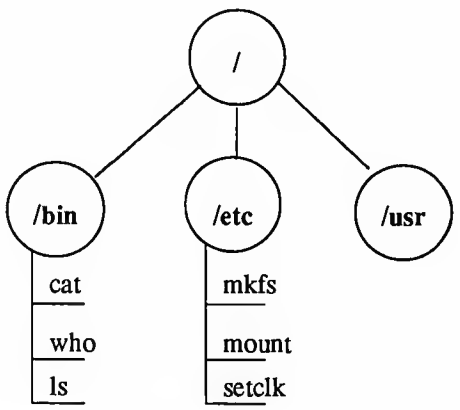


Figure 5-1: A UNIX File System

The starting point of any UNIX file system is a directory that serves as the root. In the UNIX operating system there is always one file system that is itself referred to by that name, **root**. Traditionally, the root directory of the **root** file system is represented by a single slash (/). The file system diagrammed in Figure 5-1, then, is a **root** file system. If we graft another file system onto **root** at a directory called, for example, **/usr**, the result can be illustrated by the diagram in Figure 5-2.

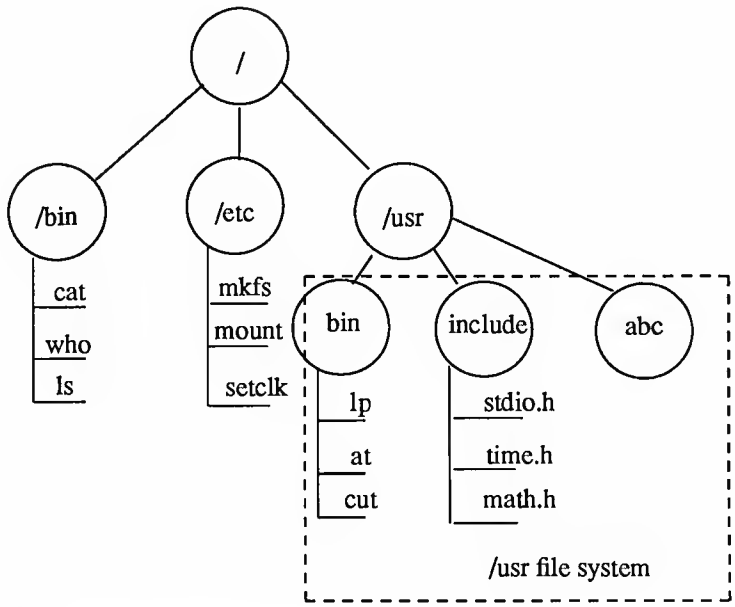


Figure 5-2: Adding the /usr File System

A directory such as /usr is referred to in various ways. You sometimes see the terms "leaf" and "mount point" used to describe a directory that is used to form the connection between the root file system and another mountable file system. Regardless of the terms used, such a directory is the root of the file system that descends from it. The name of that file system is, coincidentally, the name of the directory. In our example, the file system is /usr.

The diagrams in Figures 5-1 and 5-2 may be a convenient representation of the file and directory structure of file systems, but it is not a particularly accurate, or helpful, way of illustrating how a file system is known to the UNIX operating system. The operating system views a file system as an arrangement of addressable blocks of disk space that can be classified in four categories:

- block 0
- block 1: the super-block
- a variable number of blocks comprising the i-list
- a variable number of storage blocks: most contain data, some contain the freelist and indirect addresses

This scheme is illustrated in Figure 5-3.

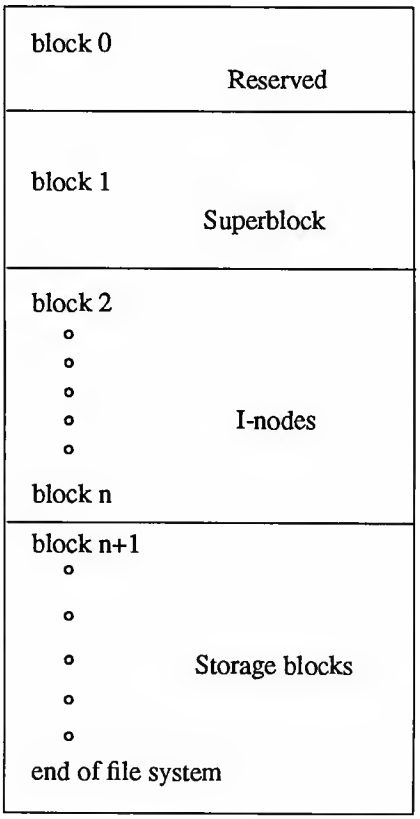


Figure 5-3: The UNIX View of a File System

Block 0

Block 0, although considered to be part of the file system, is actually not used by it. It is reserved for storing booting procedures. Not all file systems are involved in booting. For a file system that is not, block 0 is left unused.

Block 1: the Super-Block

Much of the information about the file system is stored in the super-block, including such things as

- file system size and status
  - label, file system name
  - size in physical and logical blocks
  - read-only flag
  - super-block modified flag
  - date and time of last update
- i-nodes
  - total number of i-nodes allocated
  - number of i-nodes free
  - array containing 100 free i-node numbers
  - an index into the free i-node number array
- storage blocks
  - total number of free blocks
  - array containing 50 free-block numbers
  - an index into the free-block number array

A diagram of the fields in the super-block is shown in Figure 5-4. Note that the super-block does not maintain complete lists of free i-nodes and free blocks, but only enough to meet current demands as the file system is used. At almost any time, unless the file system is close to running out of i-nodes and storage blocks, there is sure to be more free i-nodes and blocks than are listed in the super-block. The information about them is kept in one of the storage blocks.

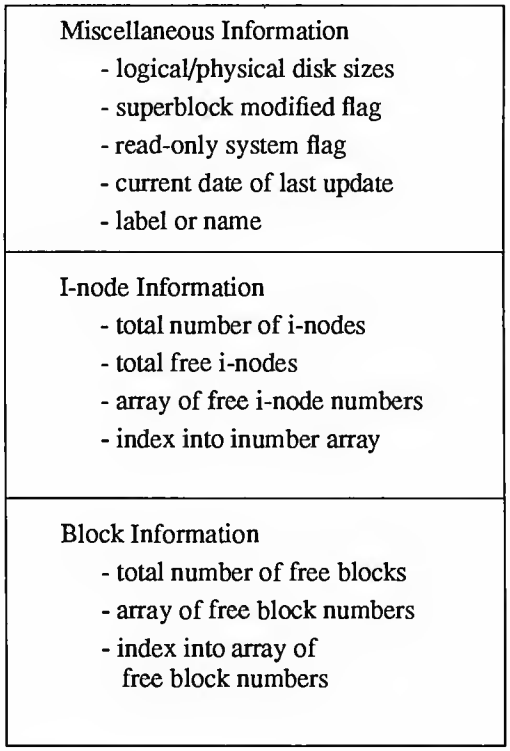


Figure 5-4: The Super-Block

I-Nodes

The term "i-node" stands for information node. (You will often see it spelled with no hyphen: inode.) The same formulation is used in other references to things associated with i-nodes. For example, the list of i-nodes is referred to as the i-list (or ilist); an i-number is the position of an i-node in the i-list.

The i-node contains all the information about a file except for its name, which is kept in a directory. An i-node is 64 bytes long, so there are 8 i-nodes to a physical block. There is no set number of blocks occupied by the i-node list; it depends on how many i-nodes are specified at the time the file system is created. An i-node contains:

- the type and mode of file: type is regular (–), directory (d), block (b), character (c), or FIFO, also known as named pipe, (p); mode is the set of read-write-execute permissions
- the number of links to the file
- the owner's user-id number
- the group-id number to which the file belongs
- the number of bytes in the file
- an array of 13 disk block addresses
- the date and time last accessed
- the date and time last modified
- the date and time created

The array of 13 disk block addresses is the heart of the i-node. The first 10 are direct addresses; that is, they point directly to the first 10 storage blocks of the contents of the file. If the file is larger than 10240 characters, the 11th address points to an indirect block that contains 256 more block addresses; the 12th address points to a double indirect block that contains the addresses of another 256 indirect blocks each of which contains the addresses of 256 storage blocks. Finally, and we're now up to files larger than 67,381,248 bytes, the 13th address in the array is the address of a triple indirect block that contains the addresses of 256 double indirect blocks, and so on. The theoretical maximum size of a UNIX system file is well beyond the limits of the amount of disk storage on your workstation.

Storage Blocks

The remainder of the space allocated to the file system is taken up by storage blocks, also called data blocks. For a regular file, the storage blocks contain the contents of the file. The contents are undefined. For a directory, the storage blocks contain 16-byte entries (64 to a block). Each entry represents a file or subdirectory that is a member of the directory. An entry consists of 2 bytes for the i-number and 14 bytes for the filename of the member file or subdirectory.

Free Blocks

Blocks not currently being used as i-nodes, as indirect address blocks, or as a storage blocks are chained together in a linked list. Each block in the list carries the address of the next block in the chain.

Summary

What we have described thus far is an abstract view of a UNIX file system, the components of a file system, and something of the way they relate to each other. In later sections of this chapter we will see hat happens to a file system when it is in use.

How the File System Works

What we have discussed so far has been the organization of a UNIX file system on paper. We now want to describe what the UNIX system does with a file system when it is being used.

Tables in Memory

When a file system is identified to the UNIX system through a **mount**(1M) command, an entry is made in the mount table, and the super-block is read into an internal buffer maintained by the kernel. Parts of the super-block that are most needed in memory are the lists of free i-nodes and free storage blocks, and the flags and time fields that are constantly being modified.

The System I-Node Table

The UNIX system maintains a structure known as the system i-node table. Whenever a file is opened its i-node is copied from the secondary storage disk into the system i-node table. If two or more processes have the same file open, they share the same i-node table entry. The entry includes, among other things:

- the name of the device from which the i-node was copied
- the i-node number or i-number
- a reference count of the number of pointers to this entry (A file can be open for more than one process.)

A diagram of the system i-node table is shown in Figure 5-5.

I-node chain pointers
Free-list chain
Flag
Waiting count for i-node
Reference count
Device where i-node resides
I-number
Mode
Number of links
User-ID of owner
Group-ID of owner
Size of file

Figure 5-5: The System I-Node Table

The System File Table

The system maintains another table called the system file table. Because files may be shared among related processes a table is needed to keep track of which files are accessible by which process. For each file descriptor, an entry in the system file table contains:

- a flag to tell how the file was opened (read/write)
- a count of the processes pointing to this entry
  (When the count drops to zero, the system drops the entry.)
- a pointer to the system i-node table
- a pointer that tells where in the file the next I/O operation will take place

The Open File Table

The last table that is used to provide access to files is the open file table. It is located in the user area portion of memory. There is a user area for each process and, consequently, an open file table for each process. An entry in the open file table contains a pointer to the appropriate system file table entry. Figure 5-6 shows how these tables point to each other.

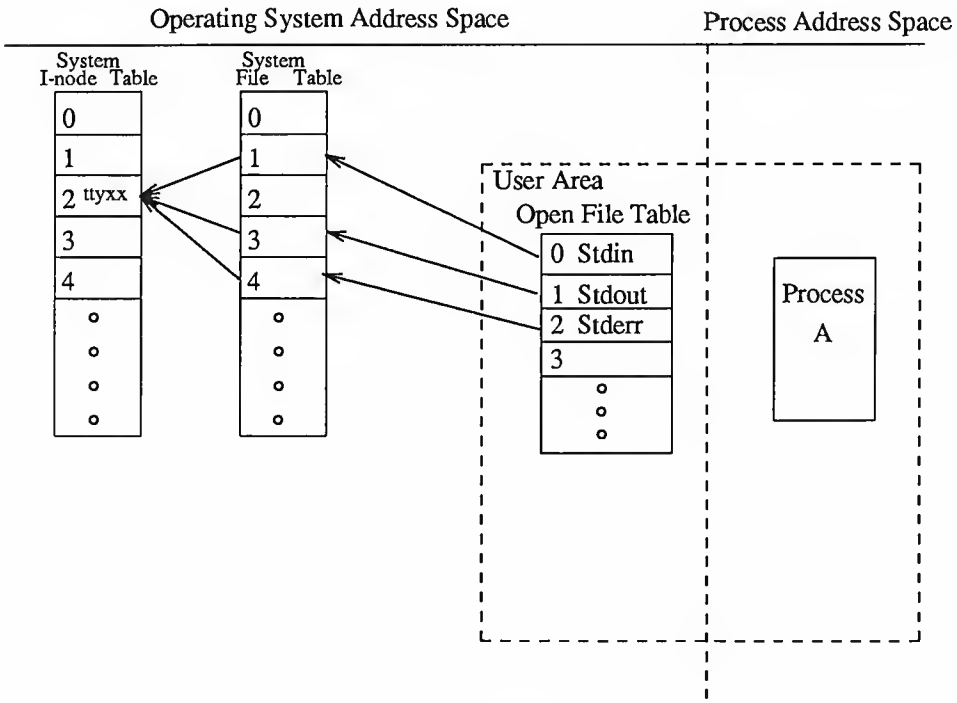


Figure 5-6: File System Tables and Their Pointers

System Steps in Accessing a File

In the next few paragraphs we describe steps followed by the operating system in opening, creating, reading, or writing a file.

Open

Suppose we give the pathname **/a/b** to the **open**(2) system call. (Our program probably uses the **fopen**(3) subroutine from the standard I/O library, but that in turn invokes the system call.)

1. The operating system sees that the pathname starts with a slash, so the root i-node is obtained from the i-node table.
2. Using the root i-node, the system does a linear scan of the root directory file looking for an entry "a". When "a" is found, the operating system picks up the i-number associated with "a".
3. The i-number gives the offset into the i-node list at which the i-node for "a" is located. At that location, the system determines that "a" is a directory.
4. Directory "a" is searched linearly until an entry "b" is found.
5. When "b" is found, its i-number is picked up and used as an index into the i-list to find the i-node for "b".
6. The i-node for "b" is determined to be a file and is copied to the system i-node table (assuming it's not already there), and the reference count is initialized.
7. The system file table entry is allocated, the pointer to the system i-node table is set, the offset for the I/O pointer is set to zero to indicate the beginning of the file, and the reference count is initialized.
8. The user area file descriptor table entry is allocated with a pointer set to the entry in the system file table.
9. The number of the file descriptor slot is returned to the program.

The linear scan algorithm for locating the i-node of a file illustrates why it is advisable to keep directories small. Search time is also speeded up by keeping subdirectory names near the beginning of a directory file. (Use **dcopy**(1M) to do this.)

Create

Creating a file (the **creat**(2) system call) has these additional steps at the beginning:

1. The super-block is referenced for a free i-node number.
2. The mode of the file is established (possibly and-ed with the complement of a **umask** entry; see **umask**(2)) and entered in the i-node.
3. Using the i-number, the system goes through a directory search similar to that used in the **open** system call. The difference is that in the case of **creat** the system writes the last portion of the pathname into the directory that is the next to last portion of the pathname. The i-number is stored with it.

Reading and Writing

Both the **read**(2) and **write**(2) system call follow these steps:

1. Using the file descriptor supplied with the call as an index, the user's open file table is read and the pointer to the system file table obtained.
2. The user buffer address and number of bytes to read(write) are supplied as arguments to the call. The correct offset into the file is read from the system file table entry.
3. (Reading) The i-node is found by following the pointer from the system file table entry to the system i-node table. The operating system copies the data from storage to the user's buffer.
4. (Writing) The same pointer chain is followed, but the system writes into the data blocks. If new direct or indirect blocks are needed, they are allocated from the file system's list of free blocks.
5. Before the system call returns to the user, the number of bytes read(written) is added to the offset in the system file table.
6. The number of bytes read or written is returned to the user.

Files Used by More Than One Process

If related processes are sharing a file descriptor (as happens after a **fork**(1)) they also share the same entry in the system file table. Unrelated processes that access the same file have separate entries in the system file table, because they may be reading from or writing to different places in the file. In both cases the entry in the i-node table is shared; the correct offset at which the read or write should take place is tracked by the offset entry in the system file table.

Pathname Conversion

The directory search and pathname conversion takes place only once as long as the file remains open. For subsequent access of the file the system supplies a file descriptor that is an index into the open file table in your user process area. The open file table points to the system file table entry where the pointer to the system i-node table is picked up. Given the i-node, the system can find the data blocks that make up the file.

Synchronization

The above description, while complex, may seem rather neat and orderly. The situation is complicated, however, by the fact that the UNIX system is a multi-tasking system. To give some tasks prompt attention, the system may make the decision that other tasks are less urgent. In addition, the system keeps a buffer cache and a cache of free blocks and i-nodes in memory together with the super-block to provide more responsive service to users. The stability that comes from having every byte of data in a file immediately written to the storage disk is traded for the gain of being able to provide more service to more users.

In normal processing, disk buffers are flushed periodically to the disk devices. This is a system process that is not related directly to any reads or writes of user processes. The process is called "synchronization." It includes writing out the super-blocks in addition to the disk buffers. The **sync** command can be used to cause the writing of super-blocks and updated i-nodes and the flushing of buffers. It is worth noting, however, that the return from the command simply means that the writing was scheduled, not necessarily completed. For this reason, many people enter the command twice in a row to introduce delay.

Search Time

There are two things that have a bearing on the amount of time the system needs to spend in looking for and reading in a file:

- the size of the directories being searched
- the size of the file itself

As described above, when the UNIX system is locating a file to be opened it searches linearly through all the directories in the pathname. Search time can be reduced in two ways:

1. Keep the number of entries in a directory low. Unless compressed with **dcopy**(1M), a directory retains it largest size. If you have a directory that has had more than 640 entries you have reached the point of indirect addresses.
2. Move subdirectory names to the start of the directory. The **dcopy**(1M) utility has an option that does this. When subdirectory names are grouped at the start of a directory the system finds a match with less searching.

Large files are slower to read because of the need to chain through indirect or double indirect addresses.

Holes in Files

When a file is created by a program, rather than by a person using an editor, the program may seek to locations in the file that are blocks away from where previous data in the file were stored. Imagine a program that has written to blocks 1 and 2 of a file. Those storage blocks are pointed to from the i-node. If the program seeks to block 5 and writes data there, the fifth address pointer in the i-node will hold a live address, but the third and fourth will still be zeros pointing to nothing. The file is now said to have a hole in it. There is nothing really serious about this; the blocks used by the file are the ones that have been written to. If the file system is reorganized (for example, by **dcopy**(1M)), storage blocks will be assigned to blocks 3 and 4; addresses will be added to the i-node; the storage blocks will be taken off the free list and initialized to zeros.

Summary

We have tried in this section to give you an understanding of how the UNIX operating system controls file systems. Seeing how things are supposed to work can give us an appreciation of the steps that must be taken to keep a file system consistent.

Administering the File System

Creating a File System and Making it Available

Once a disk is formatted the next step is to define the file system. The **mkfs**(1M) command is used for this purpose.

Using mkfs

The **mkfs** command has two formats:

mkfs special_file blocks[:i-nodes] [gap blocks/cyl]
mkfs special_file prototype [gap blocks/cyl]

Notice that in neither format is the file system actually given a name; it is identified by the filename of the special device file on which it will reside. The special device file, traditionally located in the directory **/dev**, is tied to the identifying controller and unit numbers (major and minor, respectively) for the physical device.

In the first format the only other information that must be furnished on the **mkfs** command line is the number of 512-byte blocks the file system is to occupy. The second format lets you include that information in a prototype file that can also define a directory and file structure for the new file system, and it even allows for reading in the contents of files from an existing file system.

Both formats let you specify information about the interrecord gap and the blocks per cylinder. If this information is not given on the command line, default values are used. The recommended values are different from the defaults used by the command (check Appendix A). In the first **mkfs** format, even though the number of blocks in the file is required, the number of i-nodes may be omitted. If the number of i-nodes is omitted, the command uses a default value of one i-node for every four logical storage blocks.

If you use the first format of **mkfs**, the file system is created with a single directory. If you use a prototype file, as noted above, it can include information that causes the command to build and initialize a directory and file structure for the file system. The format of a prototype file is described in the **mkfs**(1M) pages of the *IRIS-4D System Administrator's Reference Manual.*

Relating the File System Device to a File System Name

A UNIX file system is generally referred to by the name of the highest level directory in its hierarchy. The file system shown in Figure 5-2 at the beginning of this chapter is called /usr because that's the directory it is tied to. Similarly, the root file system is called that because its first directory is "root" (represented in UNIX system parlance by a slash (/)). But we saw above that when the file system was created, the only name on the command line (other than the name of a prototype file, if you used that option) was the name of a special device file. There are a couple of ways in which the file system name and the directory name can be tied together.

The first, and most explicit, is through the labelit(1M) command. **labelit** makes the connection between the device special file and the mounted name of the file system. It writes the name of the file system, that is, its highest level directory, into a field in the super-block. When **labelit** is used for removable file systems, such as those on diskette, one command line argument can be the identifying number of a volume. This number, too, is stored in a field in the super-block, but it is common practice to write it on a self-adhesive label that is attached to the diskette or tape that holds the file system.

The connection between the device and the file system name is also made by the **mount**(1M) command. This step is mandatory if the file system is to be available to users.

Mounting and Unmounting File Systems

For a file system to be available to users, the UNIX system has to be told to "mount" it. The **root** file system is always mounted as part of the boot procedure. The /usr file system, which may be the same as **root**, is also automatically mounted as the system is being brought up to multi-user mode. The issuing of the **mount** command that brings these two file systems on line is hidden in start-up shell procedures. Regardless of whether the **mount** command is hidden or not, its execution causes the file system to be listed in an internal UNIX system table (called the mount table, or /etc/mnttab) paired with the directory that is specified. For example, the command

```
# mount /dev/dsk/ips0d1s0/usr
```

tells the system that /dev/dsk/ipsa0d1s0 contains a file system that begins at directory /usr.

NOTE

The **mount** command has other arguments. See the *IRIS-4D System Administrator's Reference Manual* for complete details.

If you try to change directories (**cd**(1)) to a directory in /**usr** before the **mount** command is issued, the **cd** command will fail. Until the **mount** command completes, the system does not know about any of the directories beneath /**usr**. True, there is a directory /**usr** (it must exist at the time the **mount** command is issued), but the structure of files and directories below that remain hidden from the UNIX system until the **mount**.

Unmounting is frequently a first step before using other commands that operate on file systems. For example, **fsck**(1M), which checks and repairs a file system, and **dcopy**(1M), which copies and compresses a file system, work on unmounted file systems. Unmounting is also an important part of the process of shutting the system down.

Summary

Thus far we have looked at file systems in the abstract. We have seen something of the way they are created, stored on disks, made available to the system, or removed from the system. In the next portion of this chapter we will see how you maintain the integrity of an active file system.

Maintaining a File System

Once a file system is created and made available to users it is always necessary to monitor how it is being used by the people in the organization. We get into a somewhat fuzzy area here where the distinction between a file system and its files may result in some confusion. The administrator's view is more likely to be of the file system, while users tend to think and work in terms of files. When we begin to talk about the tasks involved in keeping file systems working smoothly for users we have to be ready to deal with users' individual files as well as with the entire system.

Once a file system has been created and made available, there are several tasks routinely done to make certain that the file systems in regular use on a workstation are providing the level of service and stability they should. They can be grouped into procedures for

- checking for file system consistency
- monitoring disk usage
- compressing and reorganizing file systems
- backing up and restoring file systems

The Need for Policies

As with most other aspects of administration, file system administration should be based on establishing a set of policies that are appropriate for your organization. There can be no hard-and-fast rules for such things as the size of file systems, the number of users in a file system, the way in which backups are done, the extent to which users can be allowed to keep inactive files in the system, or the amount of disk space a single user is entitled to occupy. These questions can only be resolved within the context of the organization. The number of users, the type of work they are doing, the number of files needed—all are variables. The responsible administrator must determine what best meets the needs of the organization.

Shell Scripts for File System Administration

Once policies have been agreed on, many of the routine tasks connected with file system administration can be incorporated in shell scripts. Monitoring disk usage, for example, can be handled through shell scripts that do the monitoring for you and transmit messages to the system console when exceptions are detected. Here are a few ideas:

- Use a shell script running under **cron**(1M) control to investigate free blocks and free i-nodes and to report on file systems that fall below a given threshold.
- Use a shell script to do automatic clean-ups of files that grow.
- Use a shell script to highlight cases of excessive use of disk space.

Checking for File System Consistency

There is a separate section later in this chapter that describes **fsck**(1M), the file system checking utility. However, we want to include this mention of it here because file system checking is central to the whole problem of normal file system maintenance.

Monitoring Disk Usage

You need to monitor the level of usage of a file system for the following reasons.

- If not watched regularly, the percentage of disk space used increases until the allocated space is used up.
- When the allocated space is used up processes run very slowly, or not at all, the system spends its time putting out a message about being out of file space.
- There is a natural tendency for users to forget about files they no longer use so that the files just sit there taking up space.
- Some files grow larger as a result of perfectly normal use of the system. It is an administrative responsibility to keep them under control.
- Some directories, notably **/tmp**, accumulate files during the day. When the system is first brought up, **/tmp** needs to have enough free blocks to carry it through to **shutdown**(1M).

There are four tasks that are part of keeping disk space uncluttered:

1. monitoring percent of disk space used
2. monitoring files and directories that grow
3. identifying and removing inactive files
4. identifying large space users

Monitoring Percent of Disk Space Used

Monitoring disk space may be done at any time to see how close to capacity your system is running. Until a pattern has emerged, it is advisable to check every day. In this example, the **df**(1M) command is used.

```
$ df -t
```

The **-t** option causes the total allocated blocks and i-nodes to be displayed, as well as free blocks and i-nodes. When no file systems are named, information about all mounted file systems is displayed.

Monitoring Files and Directories that Grow

Almost any system that is used daily has several files and directories that grow through normal use. Some examples are:

File	Use
/etc/wtmp	history of system logins
/usr/adm/sulog	history of su commands
/usr/lib/cron/log	history of actions of **/etc/cron**
/usr/lib/spell/spellhist	words that **spell**(1) fails to match

The frequency with which you should check growing files depends on how active your system is and how critical the disk space problem is. A good technique for keeping them down to a reasonable size uses a combination of **tail**(1) and **mv**(1):

```
$ tail -50 /usr/adm/sulog > /tmp/sulog

$ mv /tmp/sulog /usr/adm/sulog
```

This sequence puts the last 50 lines of /usr/adm/sulog into a temporary file, and

then it moves the temporary file to **usr/adm/sulog**, thus effectively truncating the file to the 50 most recent entries.

Identifying and Removing Inactive Files

Part of the job of cleaning up heavily loaded file systems involves locating and removing files that have not been used recently. The commands you might use to do this work are shown below; the policy decisions involved are:

- How long should a file remain unused before it becomes a candidate for removal?
- Should users be warned that old files are about to be purged?
- Should the files be permanently removed or archived?

The **find**(1) command locates files that have not been accessed recently. **find** searches a directory tree beginning at a point named on the command line. It looks for filenames that match a given set of expressions, and when a match is found, performs a specified action on the file. This example barely begins to suggest the full power of **find**.

```
$ find /usr -type f -mtime +60 -print > /tmp/deadfiles &
```

Here is what the example shows:

/usr	specifies the pathname where **find** is to start. Presumably, your machine is organized in such a way that inactive user files will not often be found in the **root** file system.
-type	**tells find** to look only for regular files, and to ignore special files, directories, and pipes.
-mtime +60	says you are interested only in files that have not been modified in 60 days.
-print	means that when a file is found that matches the **-type** and **-mtime** expressions, you want the pathname to be printed.
> /tmp/deadfiles &	directs the output to a temporary file and indicates that the process is to run in the background. This is a sensible precaution if your experience tells you to expect a substantial amount of output.

The **sysadm fileage**(1) command can be used to produce similar information.

Identifying Large Space Users

Here again the most important questions are not what commands to use to learn who is occupying excessive amounts of disk space, but rather policy questions concerned with deciding what the limits should be. Policy questions include:

- On our system, what constitutes a reasonable amount of disk space for a user to need?
- If a user exceeds the normal amount by 25%, say, is it possible the user's job requires extraordinary amounts of disk space?
- Is our system as a whole running short of space? Do our existing limits need to be reviewed?

Two commands produce useful information in this area: **du**(1) and, once again, **find**(1).

du produces a summary of the block counts for files or directories named in the command line. For example:

```
$ du /usr
```

displays the block count for all directories in the /**usr** file system. Optional arguments allow you to refine the output somewhat. For example, **du –s** may be run against each user's login to monitor individual users.

The **find** command can be used to locate specific files that exceed a given size limit.

```
$ find /usr –size +10 –print
```

This example produces a display of the pathnames of all files (and directories) in the /**usr** file system that are larger than 10 (512-byte) blocks. Similar information can be produced by the **sysadm filesize**(1) command.

File System Backup and Restore

The importance of establishing and following a file system backup plan is too often not appreciated until data are lost and cannot be recovered. Backing-up a file system takes time. Trying to recover lost or damaged data from paper records and best-guess-work takes even more time. The value of an effective system backup plan lies in the ability to recover lost or damaged data easily.

System Administration menus are provided to help you with complete and incremental file system backups to cartridge tape, and for restoring backup data to the hard disk. (For further information, see the *IRIS-4D Series Owner's Guide*, Administering File Systems.) The capability to copy selected directories and files to tape is provided by using the sysadm **store**(1) command. The directories and files are read back to the hard disk by using sysadm **restore**(1).

Another method of backing up information is to copy file systems to another computer system over a high-speed data link. The link between the machines must be a high-speed data link so that data transfers are done in a timely fashion. The other machine should be a larger system with mass storage capability.

The backup plan that you use can include any or all of these methods. The important consideration is that you evaluate the need for system backup and form a backup plan. This plan should be reevaluated as the use of the machine changes.

Complete Backup

The complete backup provided by the System Administration backup facilities copies directories and files of a specified mounted file system to diskettes or cartridge tape. Complete plus incremental backups combine to form a unified backup strategy. Complete backups can be done without intervening incremental backups, but incremental backups must be preceded by at least one complete backup to provide the base.

If the backup medium is cartridge tape, one tape can contain a file system of 45,000 (512-byte) blocks.

Incremental Backup

The incremental backup provided by the System Administration backup facilities copies files that have changed since the last backup to cartridge tape.

NOTE

Not all changed directories and files are copied in an incremental backup. The contents of **/etc/save.d/except** specifies files and directories that ARE NOT copied. A typical **/etc/save.d/except** file is shown in Figure 5-7.

The incremental file system backup is fast in comparison to the time required to do a complete backup because of the obvious fact that only files that have changed since the last prior backup (whether complete or incremental) are being collected.

```
#  Patterns of filenames to be excluded from saving by savefiles.
#  These are ed(1) regular expressions.
/.news_time$
/.yesterday$
/a.out$
/core$
/dead.answer$
/dead.letter$
/ed.hup$
/nohup.out$
/tmp/
^/etc/mnttab$
^/etc/save.d/timestamp/
^/etc/utmp$
^/etc/wtmp$
^/usr/adm/
^/usr/at/
^/usr/crash/
^/usr/dict/
^/usr/games/
^/usr/learn/
^/usr/news/
^/usr/spool/
^/usr/tmp/
```

Figure 5-7: Sample **/etc/save.d/except** File

The question of when to do a new complete backup is the critical point of a system backup plan. Some things to think about are:

- How much time do you want to devote to complete backups? Fewer tapes are required to maintain a backup library the more frequently a complete backup is done. A complete backup, however, may take 8 or 10 times as long as any single increment.

- How much time are you willing to devote to restoring a file system from backup tapes? All tapes of each incremental backup must be read to restore a file system completely. (This is rather a difficult decision to weigh properly; ideally, you will never need to restore a whole file system from backup.)
- What is the approximate rate of change in the file system under scrutiny? Do changes occur throughout the file system or in a small percentage of the files? If change is frequent and widespread, it may be best to schedule complete backups more often. If change is concentrated in a few files, perhaps selective backup should be a part of the overall plan.

Selective Backup

Use **sysadm store** to back up selected directories and files on tape. See the *IRIS-4D Series Owner's Guide*, Section 4.5, Administering File Systems.

Restoring a File System from Backup

You can restore directories and files from system backups for a single file, a directory structure, or all files of a system backup. Use the **sysadm restore**(1) command for all three types. For further information on backing up and restoring files, see the *IRIS-4D Series Owner's Guide*, Administering File Systems.

What Can Go Wrong With a File System

Most of the things that can corrupt a file system have to do with the failure of the correct pointer and count information to make it out to the storage medium. This can be caused by

- hardware failure
- program interrupts
- human error

or a combination of hardware/program failures and incorrect procedures.

Hardware Failure

There is no very effective way of predicting when hardware failure will occur. The best way of dealing with it is to be sure that recommended diagnostic and maintenance procedures are followed conscientiously. There is a utility that flags bad blocks on hard disk and uses a substitute area for blocks the system attempts to write to a flagged block, see the *IRIS-4D Series Owner's Guide*, Using the Disk Formatter/Exerciser.

Program Interrupts

It is possible that errors that cause a program to fail might result in the loss of some data. It is not easy to generalize about this because the range of possibilities is so large. Perhaps the best thing to be said is that programs should be exhaustively tested before they are put into production with valuable data.

Human Error

While it may be painful to admit it, probably the greatest cause of file system corruption falls under this heading. We are going to recommend here four rules that should be followed by anyone who manages file systems.

1. ALWAYS check a file system before mounting it. Nothing complicates the problem of cleaning up a corrupt file system so much as allowing it to be used when it is bad.
2. NEVER remove a file system physically without first unmounting it.

3. ALWAYS use the **sync** command before shutting the system down and before unmounting a file system.
4. NEVER physically write-protect a mounted file system, unless it is mounted "read only."

The random nature of all these mishaps simply underscores the importance of establishing and observing good backup practices. It is the most effective form of insurance against data loss.

Checking a File System for Consistency

When the UNIX operating system is brought up, a consistency check of the file systems should always be done. This check is automatically done as part of the power-up process. (this is still unclearXX) Included as part of that process is the command **fsstat**(1M). **fsstat** returns a code for each file system on the hard disk indicating whether the consistency checking and repair program, **fsck**(1M), should be run.

These same commands, or **sysadm checkfsys**(1), should be used to check file systems not mounted routinely as part of the power-up process. If inconsistencies are discovered, corrective action must be taken before the file systems are mounted. The remainder of this section is designed to acquaint you with the command line options of the **fsck** utility, the type of checking it does in each of its phases, and the repairs it suggests.

It should be said at the outset that file system corruption, while serious, is not all that common. Most of the time a check of the file systems finds everything all right. The reason we put so much emphasis on file system checking is that if errors are allowed to go undetected, the ultimate loss can be substantial.

The fsck Utility

The file system check (**fsck**) utility is an interactive file system check and repair program. **fsck** uses the information carried in the file system itself to perform consistency checks. If an inconsistency is detected, a message describing the inconsistency is displayed. You may elect to have **fsck** either make the repair or not. The reason you might choose to have **fsck** ignore an inconsistency is that you judge the problem to be so severe that you want either to fix it yourself using the **fsdb**(1M) utility, or you plan to go back to an earlier version of the file system. The decision to have **fsck** ignore inconsistencies and then do nothing about them yourself is not a viable one. File system inconsistencies do not repair themselves. If they are ignored, they only get worse.

The fsck Command

The **fsck** command is used to check and repair inconsistencies in a file system. With the exception of the **root** file system, a file system should be unmounted while it is being checked. The **root** file system should be checked only when the computer is in run level S and no other activity is taking place in the machine.

The following is the general format of the **fsck** command:

fsck [**–y**] [**–n**] [**–b**] [**–s***X*] [**–S***X*] [**–t***file*] [**–q**] [**–D**] [**–f**] [*fsdevice*]

The options of the **fsck** command are as follows:

–y	Specifies a "yes" response for all questions. This is the normal choice when the command is being run as part of a shell procedure. It generally causes **fsck** to correct all errors.
–n	Specifies a "no" response for all questions. **fsck** will not write the file system.
–b	Reboots the system if the file system being checked is the root (*/*) file system and changes have been made by **fsck**. If only minor, fixable damage is found, the file system is remounted.
–s*X*	Specifies an unconditional reconstruction of the free list. Following the reconstruction of the free list, the system should be rebooted so that the in-core copy of the super-block is updated (see the **–b** option). The *X* argument specifies the number of blocks-per-cylinder and the number of blocks to skip (rotational gap). The default values are those specified when the file system was created.
–S*X*	Specifies a conditional reconstruction of the free list, to be done only if corruption is detected. The format of the *X* argument is the same as described above for the **–s** option.
–t*file*	Specifies a scratch file for use in case the file system check requires additional memory. If this option is not specified, the process asks for a filename when more memory is needed.
–q	Specifies a "quiet" file system check. Output messages from the process are suppressed.
–D	Checks directories for bad blocks. This option is used to check file systems for damage after a system crash.
–f	Specifies that a fast file system check be done. Only Phase 1 (check blocks and sizes) and Phase 5 (check free list) are executed for a fast check. Phase 6 (reconstruct free list) is run only if necessary.
fsdevice	Names the special device file associated with a file system.

Sample Command Use

The command line below shows **fsck** being entered to check the **root** file system. No options are specified. The system response means that no inconsistencies were detected. The command operates in phases, some of which are run only if required or in response to a command line option. As each phase is completed, a message is displayed. At the end of the program a summary message is displayed showing the number of files (i-nodes), blocks, and free blocks.

```
# fsck /dev/dsk/ips0d1s0

  /dev/dsk/c1d0s0
  File System: root Volume: root

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  289 files 6522 blocks 3220 free
#
```

File System Components Checked by fsck

Before getting into a discussion of the **fsck** phases and the messages that may appear in each, it is important to review the components of a UNIX file system and to describe the kinds of consistency checks that are applied to them.

Super-Block

The super-block is vulnerable because every change to the file system blocks or i-nodes modifies the super-block. If the CPU is halted, and the last command involving output to the file system is not a **sync** command, the super-block is almost certainly corrupted. The super-block can be checked for inconsistencies involving:

- file system size
- i-node list size
- free-block list
- free-block count
- free i-node count

File System Size and I-Node List Size

Total file system size must be greater than the number of blocks used by the super-block plus the blocks used by the list of i-nodes. The number of i-nodes must be less than 65,500. While there is no way to check these sizes, **fsck** can check that they are within reasonable bounds. All other checks of the file system depend on the reasonableness of these values.

Free-Block List

The free-block list starts in the super-block and continues through the free-list blocks of the file system. Each free-list block can be checked for

- list count out of range
- block numbers out of range
- blocks already allocated within the file system

A check is made to see that all the blocks in the file system were found.

The first free-block list is in the super-block. The **fsck** program checks the list count for a value of less than 0 or greater than 50. It also checks each block number to make sure it is within the range bounded by the first and last data block in the file system. Each block number is compared to a list of already allocated blocks. If the free-list block pointer is not 0, the next free-list block is read and the process is repeated.

When all the blocks have been accounted for, a check is made to see if the number of blocks in the free-block list plus the number of blocks claimed by the i-nodes equals the total number of blocks in the file system. If anything is wrong with the free-block list, **fsck** can rebuild it leaving out blocks already allocated.

Free-Block Count

The super-block contains a count of the total number of free blocks within the file system. The **fsck** program compares this count to the number of blocks it found free within the file system. If the counts do not agree, **fsck** can replace the count in the super-block by the actual free-block count.

Free I-Node Count

The super-block contains a count of the number of free i-nodes within the file system. The **fsck** program compares this count to the number of i-nodes it found free within the file system. If the counts do not agree, **fsck** can replace the count in the super-block by the actual free i-node count.

I-Nodes

The list of i-nodes is checked sequentially starting with i-node 1 (there is no i-node 0). Each i-node is checked for inconsistencies involving

- format and type
- link count
- duplicate blocks
- bad block numbers
- i-node size

Format and Type

Each i-node contains a mode word. This mode word describes the type and state of the i-node. I-nodes may be one of five types:

- regular
- directory
- block special
- character special
- fifo (named pipe)

If an i-node is not one of these types, it is illegal. I-nodes may be in one of three states: unallocated, allocated, and partially allocated. This last state means an incorrectly formatted i-node. An i-node can reach this state if, for example, bad data are written into the i-node list through a hardware failure. The only corrective action **fsck** can take is to clear the i-node.

Link Count

Each i-node contains a count of the number of directory entries linked to it. The fsck program verifies the link count of each i-node by examining the total directory structure, starting from the root directory, and calculating an actual link count for each i-node.

If the link count stored in the i-node and the actual link count determined by fsck do not agree, the reason may be

- stored count not 0, actual count 0

  No directory entry appears for the i-node.
  fsck can link the disconnected file to the lost+found directory.

- stored count not 0, actual count not 0, counts unequal

  Directory entry possibly removed without i-node update.
  fsck can replace the stored link count with the actual link count.

Duplicate Blocks

Each i-node contains a list of all the blocks claimed by the i-node. The fsck program compares each block number claimed by an i-node to a list of allocated blocks. If a block number claimed by an i-node is on the allocated-blocks list, it is put on a duplicate-blocks list. If the block number is not on the allocated-blocks list, it is put there. If a duplicate-blocks list develops, fsck makes a second pass of the i-node list to find the other i-node that claims the duplicated block. While there is not enough information available to determine which i-node is in error, most of the time the i-node with the latest modification time is correct. This condition can occur by using a file system with blocks claimed by both the free-block list and by other parts of the file system.

A large number of duplicate blocks in an i-node may be caused by an indirect block not being written to the file system. The fsck program prompts the user to clear both i-nodes.

Bad Block Numbers

The fsck program checks each block number claimed by an i-node for a value lower than that of the first data block or greater than the last block in the file system. If the block number is outside this range, the block number is bad.

If there are many bad block numbers in an i-node, it may be caused by an indirect block not being written to the file system. The fsck program prompts the user to clear the i-node.

NOTE

A certain amount of semantic confusion is possible here. A bad block number in a file system is not the same as a bad (that is, unreadable) block on a hard disk.

I-Node Size

Each i-node contains a 32-bit (4-byte) size field. This size shows the number of characters in the file associated with the i-node. A directory i-node within the file system has the directory bit set in the i-node mode word. The directory size must be a multiple of 16 because a directory entry contains 16 bytes (2 bytes for the i-node number and 14 bytes for the file or directory name).

If the directory size is not a multiple of 16, **fsck** warns of directory misalignment. This is only a warning because not enough information can be gathered to correct the misalignment.

For a regular file, a rough check of the consistency of the size field of an i-node can be performed by using the number of characters shown in the size field to calculate how many blocks should be associated with the i-node, and comparing that to the actual number of blocks claimed by the i-node.

The Algorithm

The **fsck** program calculates the number of blocks that should be in a file by dividing the number of characters in an i-node by 512 (the number of characters per block) and rounding up. One block is added for each indirect block associated with the i-node.

If the actual number of blocks does not match the computed number of blocks, fsck warns of a possible file-size error. This is only a warning. Logical blocks can be created in random order, and the UNIX system does not fill them in. A check of the file would be required to tell if the error is real or not (see the section on "Holes in Files" earlier in this chapter).

Indirect Blocks

Indirect blocks are owned by an i-node. Therefore, inconsistencies in an indirect block directly affect the i-node that owns it. Inconsistencies that can be checked are:

- blocks already claimed by another i-node
- block numbers outside the range of the file system

The consistency checks described under "Duplicate Blocks" and "Bad Block Numbers" above are performed for indirect blocks as well as for the direct blocks of an i-node.

Directory Data Blocks

Directories are distinguished from regular files by an entry in the mode field of the i-node. Data blocks associated with a directory contain the directory entries. Directory data blocks are checked for inconsistencies involving:

- directory i-node numbers pointing to unallocated i-nodes
- directory i-node numbers greater than the number of i-nodes in the file system
- incorrect directory i-node numbers for "." and ".." directories
- directories disconnected from the file system

Directory Unallocated

If a directory entry i-node number points to an unallocated i-node, **fsck** can remove that directory entry. This condition occurs if the data blocks containing the directory entries are modified and written out while the i-node is not yet written out.

Bad I-Node Number

If a directory entry i-node number is pointing beyond the end of the i-node list, **fsck** can remove that directory entry. This condition occurs if bad data are written into a directory data block.

Incorrect "." and ".." Entries

The directory i-node number entry for "." should be the first entry in the directory data block. Its value should be equal to the i-node number for the directory data block.

The directory i-node number entry for ".." should be the second entry in the directory data block. Its value should be equal to the i-node number for the parent of the directory entry (or the i-node number of the directory data block if the directory is the root directory). If the directory i-node numbers for "." and ".." are incorrect, **fsck** can replace them with the correct values.

Disconnected Directories

The **fsck** program checks the general connectivity of the file system. If directories are found not to be linked into the file system, **fsck** links the directory back into the file system in the **lost+found** directory. This condition can be caused by i-nodes being written to the file system with the corresponding directory data blocks not being written to the file system. When a file is linked into the **lost+found** directory, the owner of the file needs to be told about it.

Regular Data Blocks

Data blocks associated with a regular file hold the file's contents. **fsck** does not attempt to check the validity of the contents of a regular file's data blocks.

Running fsck

The **fsck** program runs in phases. Each phase reports errors it detects. If an error is one that **fsck** can correct, the user is asked if the correction should be made. This section describes the messages that are produced by each phase.

The following abbreviations are used in the **fsck** error messages:

BLK	block number
DUP	duplicate block number
DIR	directory name
MTIME	time file was last modified
UNREF	unreferenced

The following single-letter abbreviations, used in the messages shown in the pages that follow, are replaced by the corresponding value when the message appears on your screen:

B	block number
F	file (or directory) name
I	i-node number
M	file mode
O	user-id of a file's owner
S	file size
T	time file was last modified
X	link count,
or	number of BAD, DUP, or MISSING blocks
or	number of files (depending on context)
Y	corrected link count number
or	number of blocks in file system (depending on context)
Z	number of free blocks

Figure 5-8: Error Message Abbreviations in **fsck**

Initialization Phase

Command line syntax is checked. Before the file system check can be performed, **fsck** sets up some tables and opens some files. The **fsck** program terminates on initialization errors.

General Errors

Three error messages may appear in any phase. While they seem to offer the option to continue, it is generally best to regard them as fatal, end the run, and investigate what may have caused the problem.

CAN NOT SEEK: BLK B (CONTINUE?)
The request to move to a specified block number *B* in the file system failed. The occurrence of this error condition indicates a serious problem (probably a hardware failure) that may require additional help.

CAN NOT READ: BLK B (CONTINUE?)
The request for reading a specified block number *B* in the file system failed. The occurrence of this error condition indicates a serious problem (probably a hardware failure) that may require additional help.

CAN NOT WRITE: BLK B (CONTINUE?)
The request for writing a specified block number *B* in the file system failed. The disk may be write-protected.

Meaning of Yes/No Responses

An n(no) response to the CONTINUE? prompt says:

Terminate program.
(This is the recommended response.)

A y(yes) response to the CONTINUE? prompt says:

Attempt to continue to run file system check.
Often, however, the problem persists. The error condition does not allow a complete check of the file system. A second run of **fsck** should be made to recheck this file system.

Phase 1: Check Blocks and Sizes

This phase checks the i-node list. It reports error conditions resulting from:

- checking i-node types
- setting up the zero-link-count table
- examining i-node block numbers for bad or duplicate blocks
- checking i-node size
- checking i-node format

Types of Error Messages—Phase 1

Phase 1 has three types of error messages:

1. information messages
2. messages with a CONTINUE? prompt
3. messages with a CLEAR? prompt

There is a connection between some information messages and messages with a CONTINUE? prompt. The meaning of the CONTINUE? prompt generally is that some limit of tolerance has been reached.

Meaning of Yes/No Responses—Phase 1

In Phase 1, an n(no) response to the CONTINUE? prompt says:

Terminate the program.

In Phase 1, a y(yes) response to the CONTINUE? prompt says:

Continue with the program.
This error condition means that a complete check of the file system is not possible. A second run of fsck should be made to recheck this file system.

In Phase 1, an n(no) response to the CLEAR? prompt says:

Ignore the error condition.
A NO response is only appropriate if the user intends to take other measures to fix the problem.

In Phase 1, a y(yes) response to the CLEAR? prompt says:

Deallocate i-node *I* by zeroing its contents.
This may invoke the UNALLOCATED error condition in Phase 2 for each directory entry pointing to this i-node.

Phase 1 Error Messages

UNKNOWN FILE TYPE I=I (CLEAR?)

The mode word of the i-node *I* suggests that the i-node is not a pipe, special character i-node, regular i-node, or directory i-node.

LINK COUNT TABLE OVERFLOW (CONTINUE?)

An internal table for **fsck** containing allocated i-nodes with a link count of zero has no more room.

B BAD I=I

I-node *I* contains block number *B* with a number lower than the number of the first data block in the file system or greater than the number of the last block in the file system. This error condition may invoke the EXCESSIVE BAD BLKS error condition in Phase 1 if i-node *I* has too many block numbers outside the file system range. This error condition invokes the BAD/DUP error condition in Phase 2 and Phase 4.

EXCESSIVE BAD BLOCKS I=I (CONTINUE?)

There is more than a tolerable number (usually 10) of blocks with a number lower than the number of the first data block in the file system or greater than the number of the last block in the file system associated with i-node *I*.

B DUP I=I

I-node *I* contains block number *B*, which is already claimed by another i-node. This error condition may invoke the EXCESSIVE DUP BLKS error condition in Phase 1 if i-node *I* has too many block numbers claimed by other i-nodes. This error condition invokes Phase 1B and the BAD/DUP error condition in Phase 2 and Phase 4.

EXCESSIVE DUP BLKS I=I (CONTINUE?)

There is more than a tolerable number (usually 10) of blocks claimed by other i-nodes.

DUP TABLE OVERFLOW (CONTINUE?)

An internal table in **fsck** containing duplicate block numbers has no more room.

POSSIBLE FILE SIZE ERROR I=I

The i-node *I* size does not match the actual number of blocks used by the i-node. This is only a warning. If the **-q** option is used, this message is not printed.

DIRECTORY MISALIGNED I=I
The size of a directory i-node is not a multiple of 16. This is only a warning. If the **–q** option is used, this message is not printed.

PARTIALLY ALLOCATED INODE I=I (CLEAR?)
I-node *I* is neither allocated nor unallocated.

Phase 1B: Rescan for More DUPS

When a duplicate block is found in the file system, the file system is rescanned to find the i-node that previously claimed that block. When the duplicate block is found, the following information message is printed:

B DUP I=I
I-node *I* contains block number *B*, which is already claimed by another i-node. This error condition invokes the BAD/DUP error condition in Phase 2. I-nodes with overlapping blocks may be determined by examining this error condition and the DUP error condition in Phase 1.

Phase 2: Check Path Names

This phase removes directory entries pointing to bad i-nodes found in Phase 1 and Phase 1B. It reports error conditions resulting from

- root i-node mode and status
- directory i-node pointers out of range
- directory entries pointing to bad i-nodes

Types of Error Messages—Phase 2

Phase 2 has 4 types of error messages:

1. information messages
2. messages with a FIX? prompt
3. messages with a CONTINUE? prompt
4. messages with a REMOVE? prompt

Meaning of Yes/No Responses—Phase 2

In Phase 2, an n(no) response to the FIX? prompt says:

Terminate the program since **fsck** will be unable to continue.

In Phase 2, a y(yes) response to the FIX? prompt says:

Change the root i-node type to "directory."
If the root i-node data blocks are not directory blocks, a very large number of error conditions are produced.

In Phase 2, an n(no) response to the CONTINUE? prompt says:

Terminate the program.

In Phase 2, a y(yes) response to the CONTINUE? prompt says:

Ignore DUPS/BAD error condition in root i-node and attempt to continue to run the file system check.
If root i-node is not correct, then this may result in a large number of other error conditions.

In Phase 2, an n(no) response to the REMOVE? prompt says:

Ignore the error condition.
A NO response is only appropriate if the user intends to take other measures to fix the problem.

In Phase 2, a y(yes) response to the REMOVE? prompt says:

Remove duplicate or unallocated blocks.

Phase 2 Error Messages

ROOT INODE UNALLOCATED. TERMINATING
: The root i-node (always i-node number 2) has no allocate mode bits. The occurrence of this error condition indicates a serious problem. The program stops.

ROOT INODE NOT DIRECTORY (FIX?)
: The root i-node (usually i-node number 2) is not directory i-node type.

DUPS/BAD IN ROOT INODE (CONTINUE?)
: Phase 1 or Phase 1B found duplicate blocks or bad blocks in the root i-node (usually i-node number 2) for the file system.

I OUT OF RANGE I=I NAME=F (REMOVE?)
: A directory entry *F* has an i-node number *I* that is greater than the end of the i-node list.

UNALLOCATED I=I OWNER=O MODE=M SIZE=S MTIME=T NAME=F (REMOVE?)

A directory entry *F* has an i-node *I* without allocate mode bits. The owner *O*, mode *M*, size *S*, modify time *T*, and filename *F* are printed. If the file system is not mounted and the **–n** option was not specified, the entry is removed automatically if the i-node it points to is character size 0.

DUP/BAD I=I OWNER=O MODE=M SIZE=S MTIME=T DIR=F (REMOVE?)

Phase 1 or Phase 1B found duplicate blocks or bad blocks associated with directory entry *F*, directory i-node *I*. The owner *O*, mode *M*, size *S*, modify time *T*, and directory name *F* are printed.

DUP/BAD I=I OWNER=O MODE=M SIZE=S MTIME=T FILE=F (REMOVE?)

Phase 1 or Phase 1B found duplicate blocks or bad blocks associated with file entry *F*, i-node *I*. The owner *O*, mode *M*, size *S*, modify time *T*, and filename *F* are printed.

BAD BLK B IN DIR I=I OWNER=O MODE=M SIZE=S MTIME=T

This message only occurs when the **–D** option is used. A bad block was found in DIR i-node *I*. Error conditions looked for in directory blocks are nonzero padded entries, inconsistent "." and ".." entries, and embedded slashes in the name field. This error message means that the user should at a later time either remove the directory i-node if the entire block looks bad or change (or remove) those directory entries that look bad.

Phase 3: Check Connectivity

This phase concerns itself with the directory connectivity seen in Phase 2. It reports error conditions resulting from

- unreferenced directories
- missing or full **lost+found** directories

Types of Error Messages—Phase 3

Phase 3 has 2 types of error messages:

1. information messages
2. messages with a RECONNECT? prompt

Meaning of Yes/No Responses—Phase 3

In Phase 3, an n(no) response to the RECONNECT? prompt says:

Ignore the error condition.
This invokes the UNREF error condition in Phase 4.
A NO response is only appropriate if the user intends to take other measures to fix the problem.

In Phase 3, a y(yes) response to the RECONNECT? prompt says:

Reconnect directory i-node *I* to the file system in directory for lost files (usually **lost+found**).
This may invoke a **lost+found** error condition if there are problems connecting directory i-node *I* to **lost+found**. This invokes CONNECTED information message if link was successful.

Phase 3 Error Messages

UNREF DIR I=I OWNER=O MODE=M SIZE=S MTIME=T (RECONNECT?)

The directory i-node *I* was not connected to a directory entry when the file system was traversed. The owner *O*, mode *M*, size *S*, and modify time *T* of directory i-node *I* are printed. The **fsck** program forces the reconnection of a nonempty directory.

SORRY. NO lost+found DIRECTORY

There is no **lost+found** directory in the root directory of the file system; **fsck** ignores the request to link a directory in **lost+found**. This invokes the UNREF error condition in Phase 4. Possible problem with access modes of **lost+found**.

SORRY. NO SPACE IN lost+found DIRECTORY

There is no space to add another entry to the **lost+found** directory in the root directory of the file system; **fsck** ignores the request to link a directory in **lost+found**. This invokes the UNREF error condition in Phase 4. Clean out unnecessary entries in **lost+found** or make **lost+found** larger.

DIR I=I1 CONNECTED. PARENT WAS I=I2

This is an advisory message indicating a directory i-node *I1* was successfully connected to the **lost+found** directory. The parent i-node *I2* of the directory i-node *I1* is replaced by the i-node number of the **lost+found** directory.

Phase 4: Check Reference Counts

This phase checks the link count information seen in Phases 2 and 3. It reports error conditions resulting from:

- unreferenced files
- missing or full **lost+found** directory
- incorrect link counts for files, directories, or special files
- unreferenced files and directories
- bad and duplicate blocks in files and directories
- incorrect total free-i-node counts

Types of Error Messages—Phase 4

Phase 4 has 5 types of error messages:

1. information messages
2. messages with a RECONNECT? prompt
3. messages with a CLEAR? prompt
4. messages with an ADJUST? prompt
5. messages with a FIX? prompt

Meaning of Yes/No Responses—Phase 4

In Phase 4, an n(no) response to the RECONNECT? prompt says:

Ignore this error condition.
This invokes a CLEAR error condition later in Phase 4.

In Phase 4, a y(yes) response to the RECONNECT? prompt says:

Reconnect i-node *I* to file system in the directory for lost files (usually **lost+found**).
This can cause a **lost+found** error condition in this phase if there are problems connecting i-node *I* to **lost+found**.

In Phase 4, an n(no) response to the CLEAR? prompt says:

Ignore the error condition.
A NO response is only appropriate if the user intends to take other measures to fix the problem.

In Phase 4, a y(yes) response to the CLEAR? prompt says:

Deallocate the i-node by zeroing its contents.

In Phase 4, an n(no) response to the ADJUST? prompt says:

Ignore the error condition.
A NO response is only appropriate if the user intends to take other measures to fix the problem.

In Phase 4, a y(yes) response to the ADJUST? prompt says:

Replace link count of file i-node *I* with *Y*.

In Phase 4, an n(no) response to the FIX? prompt says:

Ignore the error condition.
A NO response is only appropriate if the user intends to take other measures to fix the problem.

In Phase 4, a y(yes) response to the FIX? prompt says:

Replace count in super-block by actual count.

Phase 4 Error Messages

UNREF FILE I=I OWNER=O MODE=M SIZE=S MTIME=T (RECONNECT?)

I-node *I* was not connected to a directory entry when the file system was traversed. The owner *O*, mode *M*, size *S*, and modify time *T* of i-node *I* are printed. If the **–n** option is omitted and the file system is not mounted, empty files are cleared automatically. Nonempty files are not cleared.

SORRY. NO lost+found DIRECTORY

There is no **lost+found** directory in the root directory of the file system; **fsck** ignores the request to link a file in **lost+found**. This invokes the CLEAR error condition later in Phase 4. Possible problem with access modes of **lost+found**.

SORRY. NO SPACE IN lost+found DIRECTORY

There is no space to add another entry to the **lost+found** directory in the root directory of the file system; **fsck** ignores the request to link a file in **lost+found**. This invokes the CLEAR error condition later in Phase 4. Check size and contents of **lost+found**.

(CLEAR)
The i-node mentioned in the immediately previous UNREF error condition cannot be reconnected.

LINK COUNT FILE I=I OWNER=O MODE=M SIZE=S MTIME=T COUNT=X SHOULD BE Y (ADJUST?)
The link count for i-node *I*, which is a file, is *X* but should be *Y*. The owner *O*, mode *M*, size *S*, and modify time *T* are printed.

LINK COUNT DIR I=I OWNER=O MODE=M SIZE=S MTIME=T COUNT=X SHOULD BE Y (ADJUST?)
The link count for i-node *I*, which is a directory, is *X* but should be *Y*. The owner *O*, mode *M*, size *S*, and modify time *T* of directory i-node *I* are printed.

LINK COUNT F I=I OWNER=O MODE=M SIZE=S MTIME=T COUNT=X SHOULD BE Y (ADJUST?)
The link count for *F* i-node *I* is *X* but should be *Y*. The filename *F*, owner *O*, mode *M*, size *S*, and modify time *T* are printed.

UNREF FILE I=I OWNER=O MODE=M SIZE=S MTIME=T (CLEAR?)
I-node *I*, which is a file, was not connected to a directory entry when the file system was traversed. The owner *O*, mode *M*, size *S*, and modify time *T* of i-node *I* are printed. If the **-n** option is omitted and the file system is not mounted, empty files are cleared automatically. Nonempty directories are not cleared.

UNREF DIR I=I OWNER=O MODE=M SIZE=S MTIME=T (CLEAR?)
I-node *I*, which is a directory, was not connected to a directory entry when the file system was traversed. The owner *O*, mode *M*, size *S*, and modify time *T* of i-node *I* are printed. If the **-n** option is omitted and the file system is not mounted, empty directories are cleared automatically. Nonempty directories are not cleared.

BAD/DUP FILE I=I OWNER=O MODE=M SIZE=S MTIME=T (CLEAR?)
Phase 1 or Phase 1B found duplicate blocks or bad blocks associated with file i-node *I*. The owner *O*, mode *M*, size *S*, and modify time *T* of i-node *I* are printed.

BAD/DUP DIR I=I OWNER=O MODE=M SIZE=S MTIME=T (CLEAR?)
Phase 1 or Phase 1B found duplicate blocks or bad blocks associated with directory i-node *I*. The owner *O*, mode *M*, size *S*, and modify time *T* of i-node *I* are printed.

FREE INODE COUNT WRONG IN SUPERBLK (FIX?)
The actual count of the free i-nodes does not match the count in the super-block of the file system. If the **-q** option is specified, the count will be fixed automatically in the super-block.

Phase 5: Check Free List

This phase checks the free-block list. It reports error conditions resulting from

- bad blocks in the free-block list
- bad free-block count
- duplicate blocks in the free-block list
- unused blocks from the file system not in the free-block list
- total free-block count incorrect

Types of Error Messages—Phase 5

Phase 5 has 4 types of error messages:

1. information messages
2. messages that have a CONTINUE? prompt
3. messages that have a FIX? prompt
4. messages that have a SALVAGE? prompt

Meaning of Yes/No Responses--Phase 5

In Phase 5, an n(no) response to the CONTINUE? prompt says:

Terminate the program.

In Phase 5, a y(yes) response to the CONTINUE? prompt says:

Ignore rest of the free-block list and continue execution of **fsck**.
This error condition will always invoke BAD BLKS IN FREE LIST error condition later in Phase 5.

In Phase 5, an n(no) response to the FIX? prompt says:

Ignore the error condition.
A NO response is only appropriate if the user intends to take other measures to fix the problem.

In Phase 5, a y(yes) response to the FIX? prompt says:

Replace count in super-block by actual count.

In Phase 5, an n(no) response to the SALVAGE? prompt says:

Ignore the error condition.
A NO response is only appropriate if the user intends to take other measures to fix the problem.

In Phase 5, a y(yes) response to the SALVAGE? prompt says:

Replace actual free-block list with a new free-block list.
The new free-block list will be ordered according to the gap and cylinder specs of the –s or –S option to reduce time spent waiting for the disk to rotate into position.

Phase 5 Error Messages

EXCESSIVE BAD BLKS IN FREE LIST (CONTINUE?)
: The free-block list contains more than a tolerable number (usually 10) of blocks with a value less than the first data block in the file system or greater than the last block in the file system.

EXCESSIVE DUP BLKS IN FREE LIST (CONTINUE?)
: The free-block list contains more than a tolerable number (usually 10) of blocks claimed by i-nodes or earlier parts of the free-block list.

BAD FREEBLK COUNT
: The count of free blocks in a free-list block is greater than 50 or less than 0. This error condition will always invoke the BAD FREE LIST condition later in Phase 5.

X BAD BLKS IN FREE LIST
: *X* blocks in the free-block list have a block number lower than the first data block in the file system or greater than the last block in the file system. This error condition will always invoke the BAD FREE LIST condition later in Phase 5.

X DUP BLKS IN FREE LIST
: *X* blocks claimed by i-nodes or earlier parts of the free-list block were found in the free-block list. This error condition will always invoke the BAD FREE LIST condition later in Phase 5.

X BLK(S) MISSING
: *X* blocks unused by the file system were not found in the free-block list.

This error condition will always invoke the BAD FREE LIST condition later in Phase 5.

FREE BLK COUNT WRONG IN SUPERBLOCK (FIX?)
The actual count of free blocks does not match the count in the superblock of the file system.

BAD FREE LIST (SALVAGE?)
This message is always preceded by one or more of the Phase 5 information messages. If the **-q** option is specified, the free-block list will be salvaged automatically.

Phase 6: Salvage Free List

This phase reconstructs the free-block list. It has one possible error condition that results from bad blocks-per-cylinder and gap values.

Phase 6 Error Messages

DEFAULT FREE-BLOCK LIST SPACING ASSUMED This is an advisory message indicating the blocks-to-skip (gap) is greater than the blocks-per-cylinder, the blocks-to-skip is less than 1, the blocks-per-cylinder is less than 1, or the blocks-per-cylinder is greater than 500. The values of 7 blocks-to-skip and 400 blocks-per-cylinder are used.

You can chage these values with the -sX option on the command line. See the **fsck**(1M) and **mkfs**(1M) manual pages for further details.

Cleanup Phase

Once a file system has been checked, a few cleanup functions are performed. The cleanup phase displays advisory messages about the file system and status of the file system.

Cleanup Phase Messages

X files Y blocks Z free
This is an advisory message indicating that the file system checked contained *X* files using *Y* blocks leaving *Z* blocks free in the file system.

***** BOOT UNIX (NO SYNC!) *****
This is an advisory message indicating that a mounted file system or the root file system has been modified by **fsck**. If the UNIX system is not rebooted immediately without **sync**, the work done by **fsck** may be undone by the in-core copies of tables the UNIX system keeps. If the **-b** option of the **fsck** command was specified and the file system is **root**, a reboot is automatically done.

***** FILE SYSTEM WAS MODIFIED *****
This is an advisory message indicating that the current file system was modified by **fsck**.

Using the LP Spooler

The UNIX Line Printer (LP) Spooling Utilities is a set of eleven commands that allow you to *spool* a file that you want to print. Spooling is the name given to the technique of temporarily storing data until it is ready to be processed (in this case, by your printer). For LP spooling, a file (or group of files) to be printed is stored in a queue until a printer becomes available. When the printer is ready, the next file in the queue is printed.

LP spooling allows you to use your workstation without waiting for your file to print. LP spooling also lets you share printers among many users. The flow of printing throughout your system is regulated by the LP Spooling Utilities.

The LP Spooling Utilities allow:

- customizing your system so that it will spool to a pool of printers. (These printers need not be the same type.)
- grouping printers together into logical classes to maximize throughput.
- queueing print requests, thus allowing a print request (job) to be processed by the next available printer.
- cancelling print requests, so that an unnecessary job will not be printed.
- starting and stopping LP from processing print requests.
- changing printer configurations.
- reporting the status of the LP scheduler.
- restarting any printing that was not completed when the system was powered down.

The eleven LP spooling commands are divided into two categories: *user* commands are for general use of the LP system; *administrative* commands are for system configuration and maintenance.

Definitions and Conventions

These terms represent important concepts used in this document:

printer — A logical name that represents a physical device, i.e., the actual printer.

class — The name given to an ordered list of one or more printers. A printer may be assigned to more than one class, but need not be a member of any class.

destination The place an LP request is sent to await printing. The destination may be a specific printer or a class of printers. An output request sent to a specific printer will be printed only by that printer; a request sent to a class of printers will be printed by the first available printer in its class.

User Commands

This section describes the five basic LP commands.

User Command Summary

lp Routes jobs to a destination and places them in a queue. The destination may be either a single printer or a class of printers.

cancel Cancels output requests.

disable Prevents a printer from processing jobs in the queue.

enable Allows a printer to process jobs in the queue.

lpstat Reports the status of all aspects of the LP Spooling system.

lp : Make an Output Request

The **lp** command routes a job request to a destination where it is placed in a queue to await printing. The destination may be a single printer or a class of printers. If you do not specify a destination, the request is routed to the default destination. For information on how to set the default printer destination, see the end of this chapter.

The form of the **lp** command is:

lp [*options*] *file(s)*

Every time an **lp** request is made, a "request-ID" is assigned to the job, and a record of the request is sent to you. The request-ID has this form:

destination–seqnum

destination is the printer or class of printers to which the job has been routed. *seqnum* is an arbitrary sequence number assigned to the job by the LP system.

lp has three options which are particularly useful: **–n, –d,** and **–c** as shown in Figure 6-1.

Use **–n** to print more than one copy of a document:

lp –n*number files(s)*

number is the number of copies to print.

Use **–d** to specify a printer or class of printers other than the default printer (assuming your system is connected to more than one printer):

lp –d*destination file(s)*

Finally, use **–c** to ensure that no edits will be made to your files once you have issued a print request:

lp –c *files(s)*

You can combine these command options in any order. For a complete list of **lp** options, see the entry for **lp**(1) in the *IRIS-4D User's Reference Manual.*

```
% lp myfile
request id is myprinter-12 (1 file)
% lp < myfile
request id is myprinter-13 (standard input)
% cat myfile | lp
request id is myprinter-14 (standard input)
% lp -n3 -dfoo -c myfile
request id is foo-15 (1 file)
%
```

Figure 6-1: **lp** Command Samples

There are several different ways to request a printout with the **lp** command. The first three examples in Figure 6-1 perform identical functions, sending a simple print request to the default printer. The fourth example prints three copies on printer *foo*, and creates a copy of the file for the printer to process, thus ensuring that no changes are made to the file after the print request.

cancel : Stop a Print Request

The **cancel** command removes a job from the queue. You can **cancel** a job either before or after it has started printing, but you can **cancel** only one at a time.

Any user may cancel any other user's job. If you cancel another user's print request, mail is sent to that user. Once you **cancel** a job, you can request again only with the **lp** command.

To route a job to the printer, type:

cancel *printer-name*

cancel *request-ID*

Cancelling using the printer name cancels the job currently being printed. Using the request-ID cancels the specified job whether or not it is currently being printed as shown in Figure 6-2.

```
% cancel myprinter
request "myprinter-16" cancelled

% cancel myprinter-17
request "myprinter-17" cancelled
%
```

Figure 6-2: **cancel** Command Samples

Issuing a **cancel** command will not work when the job is being printed on a remote machine.

disable : Stop Printer from Processing Requests

The **disable** command prevents the printer from processing jobs in the queue. Possible reasons for disabling the printer include malfunctioning hardware, paper jams, running out of paper, or end-of-day shutdowns. If a printer is busy at the time it is disabled, the request it was printing is reprinted in its entirety when you re-enable the printer.

You can send job requests to a printer that has been disabled. The jobs are put on the queue but are not printed until the printer is enabled.

To **disable** a printer, type:

disable [–c] [–r*"reason"*] *printer(s)*

The **–c** option cancels the request currently being printed as well as disabling the printer. This is useful if the output is causing the printer to behave abnormally.

The **–r** option lets you tell other users why you disabled a printer. *reason* is a character string, and must be enclosed in double quotes ("). This string is reported to anyone trying to use the disabled printer.

enable : Allow Printer to Process Requests

The **enable** command permits a printer that has been disabled to begin processing jobs from the queue. To **enable** a printer, type:

enable *printer(s)*

Figure 6-3 contains examples of the **enable** command.

```
% disable -r"paper jam" myprinter
printer "myprinter" now disables
% enable myprinter
printer "myprinter" now enabled
%
```

Figure 6-3: **disable** and **enable** Command Samples

lpstat : Report LP Status

The **lpstat** command gives you a report on the status of various aspects of the LP system. To check LP status, type:

lpstat [*options*]

The most useful option is **–t**, which gives a complete report on the status of the LP system. For a complete list of options, see the entry for **lpstat**(1) in the *IRIS-4D User's Reference Manual*. Figure 6-4 contains an example of the **lpstat** command.

```
%
scheduler is running
system default destination: myprinter
members of class foo:
            myprinter
device for myprinter: /usr/spool/lp/etc/myprinter-log
myprinter accepting requests since Jul 31 21:40
foo accepting requests since Jul 30 12:23
printer myprinter now printing foo-18
endbled since Aug 5 15:34
foo-18 mylogin 3156 Aug 7 17:11on myprinter
%
```

Figure 6-4: **lpstat** Command Samples

Administrative Commands

This section summarizes the commands that are used to administer the LP system. To execute the administrative commands, you must be logged in as either *root* (i.e., as the super-user) or as lp. Inexperienced users should not use the LP administrative commands.

Administrative Command Summary

lpsched Starts the LP scheduler.

lpshut Stops the LP scheduler.

reject Prevents jobs from being queued at a particular destination.

accept Permits job requests to be queued at a particular destination.

lpmove Moves printer requests from one destination to another.

lpadmin Configures the LP system.

lpsched : Start the LP Scheduler

The **lpsched** command starts the LP scheduler. LP prints jobs only when the scheduler is running. **lpsched** is executed automatically each time the IRIS is booted.

Every time **lpsched** is executed, it creates a file called *SCHEDLOCK* in **/usr/spool/lp**. As long as this file exists, the system will not allow another scheduler to run. When the scheduler is stopped under normal conditions, *SCHEDLOCK* is automatically removed. If the scheduler stops abnormally, you must remove *SCHEDLOCK* before you use the **lpsched** command. This procedure may also be necessary to restart the scheduler after the system shuts down abnormally.

To start the LP scheduler, type:

```
/usr/lib/lpsched
```

There is no response from the system to acknowledge the **lpsched** command; to verify that the scheduler is running, use **lpstat**.

lpshut : Stop the LP Scheduler

The **lpshut** command stops the LP scheduler and ends all printing activity. All requests that are being printed when you issue the **lpshut** command are reprinted in their entirety when the scheduler is restarted.

To stop the LP scheduler, type:

```
/usr/lib/lpshut
```

reject : Prevent Print Requests

The **reject** command stops **lp** from routing requests to a destination queue. For example, if a printer has been removed for repairs, or has received too many requests, you may wish to prevent new jobs from being queued at that destination.

All requests that are in the queue when you issue the **reject** command are printed if the printer is enabled.

The **reject** command takes the form:

```
/usr/lib/reject [-r"reason"] destination(s)
```

The **-r** option lets you tell other users why print requests are being rejected. *reason* is a character string, and is enclosed in double quotes ("). This string is reported to anyone trying to use **lp** to send requests to the specified destination.

accept : Allow Print Requests

The **accept** command allows job requests to be placed in a queue at the named printer(s) or class(es) of printers. To let a printer receive job requests, type:

/usr/lib/accept *destination(s)*

as shown in Figure 6-5.

```
% su

# usr/lib/accept myprinter
destination "myprinter" now accepting requests

# /usr/lib/reject -r"printer broken" myprinter
destination "myprinter" is no longer accepting requests
```

Figure 6-5: **reject** and **accept** Command Samples

lpmove : Move a Request to Another Printer

The **lpmove** command moves print requests from one destination to another. For example, if you have a printer removed for repairs, you may want to move all jobs pending on the queue to a destination with a working printer. You may also use **lpmove** to move specific requests from one destination to another, but only after you have halted the scheduler with the **lpshut** command. **lpmove** automatically rejects job requests re-routed to a destination without a printer.

The **lpmove** command takes two forms:

/usr/lib/lpmove *dest1 dest2*

/usr/lib/lpmove *request(s) destination*

dest1, *dest2*, and *destination* are printers or classes of printers. *request* is a specific request-ID.

In the first form of the command, all requests are moved from *dest1* to *dest2*. After the move, the printer or printers at *dest1* will not accept requests until you issue an **accept** command. All re-routed requests are renamed *dest2–nnn*, where *nnn* is a new sequence number in the queue for destination *dest2*.

In the second form, which you may issue only after you stop the scheduler, the re-routed requests are renamed *destination–nnn*. When you restart the scheduler, the original destinations will still accept new requests. Figure 6-6 contains examples of the **lpmove** command.

```
% su
# /usr/lib/lpmove myprinter yourprinter
# /usr/lib/lpmove foo-19 foo-20 yourprinter
total of 2 requests moved to yourprinter
#
```

Figure 6-6: **lpmove** Command Samples

lpadmin : Configure Printers

The **lpadmin** command has two primary uses: adding new printers to the system, and changing printer classes and destinations. Since Silicon Graphics supplies routines to automatically add the printers supported for use with the IRIS, the options for adding printers are useful only in the case of dumb printers. These options are covered in the chapter called, Attaching a Terminal, Modem, or Dumb Printer in the *IRIS-4D Series Owner's Guide*.

Unlike most UNIX commands, **lpadmin** requires an option. The **lpadmin** command takes three forms:

lpadmin –**d**[*destination*]

lpadmin –**x***destination*

lpadmin –**p***printer*

The –**d** option sets the system default destination. The *destination* must already exist when you issue the command. For complete instructions on how to define the default destination, see the end of this chapter.

The **–x** option removes the specified *destination* from the LP system. This form of the **lpadmin** command will NOT work while the scheduler is running.

You cannot remove a destination (printer or class) if it has pending requests; you must first either remove all requests with the **cancel** command or move them to other destinations with **lpmove**.

Removing the last remaining member of a class deletes that class from LP. Removal of a class, however, does not imply the removal of printers assigned to that class.

The **–p** form of the **lpadmin** command has two options that let you re-assign printers to different classes as shown in Figure 6-7. With these options, the **lpadmin** command takes the form:

lpadmin -p*printer* [–c*class*] [–r*class*]

The **–c** option assigns a *printer* to the specified *class*; the **–r** option removes a *printer* from the specified *class*.

```
% su
# /usr/lib/lpadmin -xmyprinter
# /usr/lib/lpadmin -dmyprinter -rfoo -cboo
#
```

Figure 6-7: **lpadmin** Command Samples

The **-p** options will not work while the scheduler is running. For a complete list of options, see the entry for **lpadmin** (1M) in the *IRIS-4D System Administrator's Reference Manual.*

Changing the Default Printer Destination

The **lp** command determines the destination of a request by checking for a **–d** option on the command line. If no **–d** is present, it checks to see if the environment variable LPDEST is set. If LPDEST is not set, then the request is routed to the default destination.

The system default destination can be a printer or a printer class. It is set by using the **lpadmin** command with the **–d** option. The system default must be set by the user. A destination must already exist on the LP system before you can designate it as the default destination.

Setting the environment variable LPDEST allows a user to have a default destination other than the system default.

Figure 6-8 illustrates examples of setting the system default with **lpadmin** and setting the user default with LPDEST.

```
% su
# /usr/lib/lpadmin -dmyprinter
#
% setenv LPDEST yourprinter
%
```

Figure 6-8: Setting the Default Printer

TTY

This chapter covers the following topics:

- The terms used in discussing TTY management
- How the TTY system works
- How to tell what line settings are defined
- How to create new line settings and hunt sequences
- How to modify TTY line characteristics
- How to set terminal options

Definition of Terms

The following terms are used in this chapter:

TTY	Derived from the near-classic abbreviation for teletype-writer, the term covers the whole area of access between the UNIX system and peripheral devices, including the system console. It shows up in commands such as **getty**(1M) and **stty**(1), in the names of device special files such as **/dev/tty01**, and in the names of files such as **/etc/gettydefs**, which is used by **getty**.
TTY line	The physical equipment through which access to the computer is made.
port	A synonym for TTY line.
line settings	A set of line characteristics.
baud rate	The speed at which data is transmitted over the line. A part of line settings.
mode	The characteristics of the terminal interface. A part of line settings. The TTY line and the terminal must be working in the same mode before communication can take place. Described in **termio**(7).

hunt sequence	A circular series of line settings such as different baud rates. During the login sequence, a user looking for a compatible connection to the computer can go from one setting to the next by sending a BREAK signal.
terminal options	Selectable settings that define the way a given terminal operates. Described in **termio**(7).

The TTY System

The remaining sections in this chapter describe how the TTY system operates, and how you can administer it.

How the TTY System Works

A series of four processes (**init**(1M), **getty**(1M), login(1), **sh**(1), or **csh**(1)) connects a user to the UNIX system. **init** is a general process spawner that is invoked as the last step in the boot procedure. It spawns a **getty** process for each line that a user may log in on, guided by instructions in **/etc/inittab**. An argument required by the **getty** command is **line**. The TTY line argument is the name of a special file in the **/dev** directory. For a description of other arguments that may be used with **getty** see the *IRIS-4D System Administrator's Reference Manual.*

A user attempting to make a connection generates a request-to-send signal that is routed by the hardware to the **getty** process for one of the TTY line files in **/dev**. (We're omitting how the signal gets from the user's terminal to the workstation.) **getty** responds by sending an entry from file **/etc/gettydefs** down the line. The **gettydefs** entry used depends on the **speed** argument used with the **getty** command. (In the SYNOPSIS of the **getty**(1M) command the argument name is **speed**, but it is really a pointer to the **label** field of a **gettydefs** entry.) If no **speed** argument is provided, **getty** uses the first entry in **gettydefs**. Among the fields in the **gettydefs** entry (described later in this chapter) is the login prompt.

On receiving the login prompt, the user enters a login name. **getty** starts **login**, using the login name as an argument. **login** issues the prompt for a password, evaluates the user's response, and assuming the password is acceptable, calls in the user's shell as listed in the **/etc/passwd** entry for the login name. If no shell is named, **/bin/sh** is furnished by default. **login** also executes **/etc/profile.**

/bin/sh executes the user's **.profile**, if it exists. **.profile** or **.login** often contain **stty** commands that reset terminal options that differ from the defaults. The connection between the user and the UNIX system has now been made.

How to Tell What Line Settings Are Defined

You have two ways to check line settings.

1. Through the System Administration Menus, specifically the **sysadm**(1) **lineset** subcommand. **sysadm lineset** first shows the full range of line settings, then gives you the chance to examine a line in detail.

2. By looking directly in **/etc/gettydefs**.

The **/etc/gettydefs** file contains information used by the **getty**(1M) command to establish the speed and terminal settings for a line. The general format of the **gettydefs** file is:

```
label# initial-flags # final-flags #login-prompt #next-label
```

Figure 7-1 shows a few lines from a **gettydefs** file.

```
19200# B19200 HUPCL # B19200 SANE IXANY TAB3 HUPCL #login: #9600

9600# B9600 HUPCL # B9600 SANE IXANY TAB3 HUPCL #login: #4800

4800# B4800 HUPCL # B4800 SANE IXANY TAB3 HUPCL #login: #2400

2400# B2400 HUPCL # B2400 SANE IXANY TAB3 HUPCL #login: #1200

1200# B1200 HUPCL # B1200 SANE IXANY TAB3 HUPCL #login: #300

300# B300 HUPCL # B300 SANE IXANY TAB3 HUPCL #login: #19200
```

Figure 7-1: **gettydefs** Entries

The entries shown in Figure 7-1 form a single, circular hunt sequence; the last field on each line is the label of the next line. The next-label field for the last line shown points back to the first line in the sequence. The object of the hunt sequence is to link a range of line speeds. If you see garbage characters instead of a clear login prompt, entering a BREAK causes **getty** to step to the next entry in the sequence. The hunt continues until the baud rate of the line matches the speed of the user's terminal. The flag fields shown have the following meanings:

B300-B19200	The baud rate of the line.
HUPCL	Hang up on close.
SANE	A composite flag that stands for a set of normal line characteristics.

IXANY	Allow any character to restart output. If this flag is not specified, only DC1 (CTL-Q) will restart output.
TAB3	Send tabs to the terminal as spaces.

For a description of all **getty** flags, see **termio**(7).

How to Create New Line Settings and Hunt Sequences

You have two ways to do this.

1. Use the System Administration Menus, specifically the **sysadm mklineset**(1) subcommand. **sysadm mklineset** leads you through a series of prompts. Your responses make up the information for a new **gettydefs** entry.
2. By using **ed**(1) or **vi**(1) to edit **/etc/gettydefs**.

Create new lines for the **gettydefs** file by following the example shown above. Each entry in the file is followed by a blank line. After editing the file run the command:

```
# /etc/getty –c /etc/gettydefs
```

This causes **getty** to scan the file and print the results on your terminal. If there are any unrecognized modes or improperly constructed entries, they are reported.

How to Modify TTY Line Characteristics

You have two ways to do modify TTY line characteristics.

1. Use the System Administration Menus, specifically the **sysadm modtty**(1) subcommand. **sysadm modtty** leads you through a series of prompts. Your responses edit a "getty" entry in **/etc/inittab**.
2. By using **ed**(1) or **vi**(1) to edit **/etc/inittab**.

The **/etc/inittab** file contains instructions for the **/etc/init**(1M) command. The general format of a line entry in the **/etc/inittab** file is as follows.

identification:level:action:process

The four colon-separated fields are as follows.

identification	A unique one- or two-character identifier for the line entry.
level	The run-level in which the entry is to be performed.
action	How **/etc/init** treats the process field (refer to the **inittab**(4) manual page for complete information).
process	The shell command to be executed.

/etc/inittab contains several entries that spawn **getty** processes. Figure 7-2 is a selection of such entries **grep**'ped from a sample **/etc/inittab**.

```
co:1234:respawn:/etc/getty console console
ct:2:off:/etc/getty contty contty;#line not in use
21:2:respawn:/etc/getty tty21 9600
22:2:respawn:/etc/getty tty22 9600
23:2:respawn:/etc/getty tty23 9600
24:2:off:/etc/getty tty24 9600;#line not in use
25:2:off:/etc/getty tty25 9600;#line not in use
```

Figure 7-2: **getty** Entries from **/etc/inittab**

There are at least three things you might want to do to an **inittab** entry for a TTY line:

1. Change the action. Two actions that apply to TTY lines are "respawn" and "off" (see the **inittab**(4) manual page for complete information on this field).
2. Add or change arguments to **/etc/getty** in the process field. A frequently used argument is **–t***nn*. This tells **getty** to hang up if nothing is received within *nn* seconds. It's good practice to use the **–t** argument on dial-up lines.
3. Add or change comments. Comments can be inserted after a semi-colon (;) to end the command, and a pound sign (#) to start the comments.

How to Set Terminal Options

The TTY system described thus far establishes a basic style of communication between the user's terminal and the UNIX operating system. Once the user has successfully logged in, there may be terminal options that would be preferable to ones in the default set.

The command that is used to control terminal options is **stty**(1). Many users add an **stty** command to their **.profile** so the options they want are automatically set as part of the **login** process. Here is an example of a simple **stty** command.

```
$ stty cr0 nl0 echoe –tabs erase ^H
```

The options in the example mean:

cr0 nl0	No delay for carriage return or new line. Delays are not used on a video display terminal, but are necessary on some printing terminals to allow time for the mechanical parts of the equipment to move.
echoe	Erases characters as you backspace.
–tabs	Expand tabs to spaces when printing.
erase ^H	Change the character-delete character to a ^H. The default character-delete character is the pound sign (#). Most terminals transmit a ^H when the **backspace** key is pressed. Specifying this option makes the **backspace** key useful.

Basic Networking Utilities

The Basic Networking Utilities let computers using the UNIX operating system communicate with each other and with remote terminals. These utilities range from those used to copy files between computers (**uucp** and **uuto**) to those used for remote login and command execution (**cu**, **ct**, and **uux**).

As an administrator, you need to be familiar with the administrative tools, logs, and database files used by the Basic Networking Utilities. This chapter goes into detail about the Basic Networking Utilities files, directories, daemons, and commands.

Networking Hardware

Before your computer can communicate with other computers, you must set up the hardware to complete the communications link. The cables and other hardware you will need depend on how you want to connect the computers: direct links, telephone lines, or local area networks.

Direct Links

You can create a direct link to another computer by running cables between serial ports on the two computers. Direct links are useful where two computers communicate regularly and are physically close—within 50 feet of each other. You can use a limited distance modem to increase this distance somewhat. Transfer rates of up to 19200 bits per second (bps) are possible when computers are directly linked.

Telephone Lines

Using an Automatic Call Unit (ACU), your computer can communicate with other computers over standard phone lines. The ACU dials the telephone number requested by the networking utilities. The computer it is trying to contact must have a telephone modem capable of answering incoming calls.

Local Area Network

A Local Area Network (LAN) can be the communication medium for basic networking. Once your computer is established as a node on a LAN, it will be able to contact any other computer connected to the LAN.

Networking Commands

Basic networking programs can be divided into two categories: user programs and administrative programs. The following paragraphs describe the programs in each category.

User Programs

The user programs for basic networking are in **/usr/bin**. No special permission is needed to use these programs. These commands are all described in the *IRIS-4D User's Reference Manual*.

cu	Connects your computer to a remote computer so you can be logged in on both at the same time, allowing you to transfer files or execute commands on either computer without dropping the initial link.
ct	Connects your computer to a remote terminal so the user of the remote terminal can log in. The user of a remote terminal can call the computer and request that the computer call it back. In this case, the computer drops the initial link so that the remote terminal's modem will be available when it is called back.
uucp	Lets a user copy a file from one computer to another. It creates work files and data files, queues the job for transfer, and calls the **uucico** daemon, which in turn attempts to contact the remote computer.
uuto	Copies files from one computer to a public spool directory on another computer (**/usr/spool/uucppublic/receive**). Unlike **uucp**, which lets you copy a file to any accessible directory on the remote computer, **uuto** places the file in an appropriate spool directory and tells the remote user to pick it up with **uupick**.
uupick	Retrieves the files placed under **/usr/spool/uucppublic/receive** when files are transferred to a computer using **uuto**.
uux	Creates the work, data, and execute files needed to execute commands on a remote computer. The work file contains the same information as work files created by **uucp** and **uuto**. The execute files contain the command string to be executed on the remote computer and a list of the data files. The data files are those files required for the command execution.

uustat	Displays the status of requested transfers (**uucp**, **uuto**, or **uux**). It also provides you with a means of controlling queued transfers.

Administrative Programs

Most of the administrative programs are in /usr/lib/uucp, along with basic networking database files and shell scripts. The only exception is **uulog**, which is in /usr/bin. These commands are described in the *IRIS-4D System Administrator's Reference Manual*.

You should use the **uucp** login ID when you administer the Basic Networking Utilities because it owns the basic networking and spooled data files. The home directory of the **uucp** login ID is /usr/lib/uucp. (The other basic networking login ID is **nuucp**, used by remote computers to access your computer. Calls from **nuucp** are answered by **uucico**.)

uulog	Displays the contents of a specified computer's log files. Log files are created for each remote computer your computer communicates with. The log files contain records of each use of **uucp**, **uuto**, and **uux**.
uucleanup	Cleans up the spool directory. It is normally executed from a shell script called **uudemon.cleanup**, which is started by **cron**.
Uutry	Tests call processing capabilities and does a moderate amount of debugging. It invokes the **uucico** daemon to establish a communication link between your computer and the remote computer you specify.
uucheck	Checks for the presence of basic networking directories, programs, and support files. It can also check certain parts of the **Permissions** file for obvious syntactic errors.

Daemons

There are three daemons in the Basic Networking Utilities. A daemon is a routine that runs as a background process and performs a system-wide public function. These daemons handle file transfers and command executions. They can also be run manually from the shell.

uucico Selects the device used for the link, establishes the link to the remote computer, performs the required login sequence and permission checks, transfers data and execute files, logs results, and notifies the user by **mail** of transfer completions. When the local **uucico** daemon calls a remote computer, it "talks" to the **uucico** daemon on the remote computer during the session.

The **uucico** daemon is executed by **uucp**, **uuto**, and **uux** programs, after all the required files have been created, to contact the remote computer. It is also executed by the **uusched** and **Uutry** programs.

uuxqt Executes remote execution requests. It searches the spool directory for execute files (always named X.**file**) that have been sent from a remote computer. When an X.**file** file is found, **uuxqt** opens it to get the list of data files that are required for the execution. It then checks to see if the required data files are available and accessible. If the files are present and can be accessed, **uuxqt** checks the **Permissions** file to verify that it has permission to execute the requested command. The **uuxqt** daemon is executed by the **uudemon.hour** shell script, which is started by **cron**.

uusched Schedules the queued work in the spool directory. Before starting the **uucico** daemon, **uusched** randomizes the order in which remote computers will be called. **uusched** is executed by a shell script called **uudemon.hour**, which is started by **cron**.

Internal Programs

uugetty This program is very similar to the **getty** program except it permits a line (port) to be used in both directions. A **uugetty** will be assigned to a port in the **/etc/inittab** file if bidirectional is chosen when you modify a port using the **sysadm**(1) **portmgmt** command. **uugetty** is executed as a function of the **init** program and is described in the *IRIS-4D System Administrator's Reference Manual.*

Supporting Data Base

The Basic Networking Utilities support files are in the /usr/lib/uucp directory. Most changes to these files can be made using the System Administration Menu commands. The descriptions below, however, provide details on the structure of these files so you can edit them manually.

Devices	Contains information concerning the location and line speed of the automatic call unit, direct links, and network devices.
Dialers	Contains character strings required to negotiate with network devices (automatic calling devices) in the establishment of connections to remote computers (non 801-type dialers).
Systems	Contains information needed by the **uucico** daemon and the **cu** program to establish a link to a remote computer. It contains information such as the name of the remote computer, the name of the connecting device associated with the remote computer, when the computer can be reached, telephone number, login ID, and password.
Dialcodes	This file contains dial-code abbreviations that may be used in the phone number field of **Systems** file entries.
Permissions	This file defines the level of access that is granted to computers when they attempt to transfer files or remotely execute commands on your computer.
Poll	This file defines computers that are to be polled by your system and when they are polled.
Sysfiles	This file is used to assign different or multiple files to be used by **uucico** and **cu** as **Systems**, **Devices**, and **Dialers** files.

Devices File

The **Devices** file (**/usr/lib/uucp/Devices**) contains information for all the devices that may be used to establish a link to a remote computer, devices such as automatic call units, direct links, and network connections.

NOTE

This file works closely with the **Dialers**, **Systems**, and **Dialcodes** files. Before you make changes in any of these files, you should be familiar with them all. A change to an entry in one file may require a change to a related entry in another file.

Each entry in the **Devices** file has the following format:

Type Line Line2 Class Dialer-Token-Pairs

Each of these fields is defined in the following section.

Type This field may contain one of two keywords (**Direct** or **ACU**), the name of a Local Area Network switch, or a system name.

Direct This keyword indicates a Direct Link to another computer or a switch (for **cu** connections only).

ACU This keyword indicates that the link to a remote computer is made through an automatic call unit (Automatic Dial Modem). This modem may be connected either directly to your computer or indirectly through a Local Area Network (LAN) switch.

LAN_Switch
This value can be replaced by the name of a LAN switch. **micom** and **develcon** are the only ones for which there are caller scripts in the **Dialers** file. You can add your own LAN switch entries to the **Dialers** file.

Sys-Name
This value indicates a direct link to a particular computer. (**Sys-Name** is replaced by the name of the computer.) This naming scheme is used to convey the fact that the line associated with this **Devices** entry is for a particular computer in the **Systems** file.

The keyword used in the **Type** field is matched against the third field of **Systems** file entries as shown below:

Devices: `ACU tty11 - 1200 penril`

Systems:
```
eagle Any ACU 1200 3251 ogin: nuucp \
        ssword: Oakgrass
```

You can designate a protocol to use for a device within this field. See the Protocols section at the end of the description of this file.

Line This field contains the device name of the line (port) associated with the **Devices** entry. For instance, if the Automatic Dial Modem for a particular entry was attached to the **/dev/tty11** line, the name entered in this field would be **tty11**.

Line2 If the keyword **ACU** was used in the **Type** field and the ACU is an 801 type dialer, **Line2** would contain the device name of the 801 dialer. (801 type ACUs do not contain a modem. Therefore, a separate modem is required and would be connected to a different line, defined in the **Line** field.) This means that one line would be allocated to the modem and another to the dialer. Since non-801 dialers will not normally use this configuration, the **Line2** field will be ignored by them, but it must still contain a hyphen (-) as a placeholder.

Class If the keyword **ACU** or **Direct** is used in the **Type** field, **Class** may be just the speed of the device. However, it may contain a letter and a speed (for example, C1200, D1200) to differentiate between classes of dialers (Centrex or Dimension PBX). This is necessary because many larger offices may have more than one type of telephone network: one network may be dedicated to serving only internal office communications while another handles the external communications. In such a case, it becomes necessary to distinguish which line(s) should be used for internal communications and which should be used for external communications. The keyword used in the **Class** field of the **Devices** file is matched against the fourth field of **Systems** file entries as shown below:

Devices: `ACU tty11 - D1200 penril`

Systems:
```
eagle Any ACU D1200 3251 ogin: nuucp \
        ssword: Oakgrass
```

Some devices can be used at any speed, so the keyword **Any** may be used in the **Class** field. If **Any** is used, the line will match any speed requested in a **Systems** file entry. If this field is **Any** and the **Systems** file **Class** field is **Any**, the speed defaults to 1200 bps.

Dialer-Token-Pairs:
This field contains pairs of dialers and tokens. The **dialer** portion may be the name of an automatic dial modem, a LAN switch, or it may be **direct** for a Direct Link device. You can have any number of Dialer-Token-Pairs. The **token** portion may be supplied immediately following the **dialer** portion or if not present, it will be taken from a related entry in the **Systems** file.

This field has the format:

dialer token dialer token

where the last pair may or may not be present, depending on the associated device (dialer). In most cases, the last pair contains only a **dialer** portion and the **token** portion is retrieved from the **Phone** field of the **Systems** file entry. A valid entry in the **dialer** portion may be defined in the **Dialers** file.

The **Dialer-Token-Pairs (DTP)** field may be structured four different ways, depending on the device associated with the entry:

1. If an automatic dialing modem is connected directly to a port on your computer, the **DTP** field of the associated **Devices** file entry will only have one pair. This pair would normally be the name of the modem. This name is used to match the particular **Devices** file entry with an entry in the **Dialers** file. Therefore, the **dialer** field must match the first field of a **Dialers** file entry as shown below:

   Devices: `ACU ttyl1 - 1200` **ventel**

   Dialers: **ventel** `=&-% "" \r\p\r\c $ <K\T%%\r>\c ONLINE!`

   Notice that only the **dialer** portion (**ventel**) is present in the **DTP** field of the **Devices** file entry. This means that the **token** to be passed on to the dialer (in this case the phone number) is taken from the **Phone** field of a **Systems** file entry. (\T is implied, see below.) Backslash sequences are described below.

2. If a direct link is established to a particular computer, the **DTP** field of the associated entry would contain the keyword **direct**. This is true for both types of direct link entries, **Direct** and **System-Name** (refer to discussion on the **Type** field).

3. If a computer with which you wish to communicate is on the same local network switch as your computer, your computer must first access the switch and the switch can make the connection to the other computer. In this type of entry, there is only one pair. The **dialer** portion is used to match a **Dialers** file entry as shown below:

   Devices: `develcon ttyl3 - 1200` **develcon** `\D`

   Dialers: **develcon** `"" "" \pr\ps\c est:\007 \E\D\e \007`

As shown, the **token** portion is left blank, which indicates that it is retrieved from the **Systems** file. The **Systems** file entry for this particular computer will contain the token in the **Phone** field, which is normally reserved for the phone number of the computer (refer to **Systems** file, **Phone** field). This type of **DTP** contains an escape character (\D), which ensures that the contents of the **Phone** field will not be interpreted as a valid entry in the **Dialcodes** file.

4. If an automatic dialing modem is connected to a switch, your computer must first access the switch and the switch will make the connection to the automatic dialing modem. This type of entry requires two **dialer-token-pairs**. The **dialer** portion of each pair (fifth and seventh fields of entry) will be used to match entries in the **Dialers** file as shown below:

```
Devices:  ACU tty14 - 1200 develcon vent ventel

Dialers:  develcon "" "" \pr\ps\c est:\007 \E\D\e \007
Dialers:  ventel =&-% "" \r\p\r\c $ <K\T%%\r>\c ONLINE!
```

In the first pair, **develcon** is the dialer and **vent** is the token that is passed to the Develcon switch to tell it which device (ventel modem) to connect to your computer. This token would be unique for each LAN switch since each switch may be set up differently. Once the ventel modem has been connected, the second pair is accessed, where ventel is the dialer and the token is retrieved from the **Systems** file.

There are two escape characters that may appear in a **DTP** field:

\T Indicates that the **Phone (token)** field should be translated using the **Dialcodes** file. This escape character is normally placed in the **Dialers** file for each caller script associated with an automatic dial modem (penril, ventel, etc.). Therefore, the translation will not take place until the caller script is accessed.

\D Indicates that the **Phone (token)** field should not be translated using the **Dialcodes** file. If no escape character is specified at the end of a **Devices** entry, the **\D** is assumed (default). A **\D** is also used in the **2ialers** file with entries associated with network switches (develcon and micom).

Protocols

You can define the protocol to use with each device. In most cases it is not needed since you can use the default or define the protocol with the particular System you are calling (see **Systems** file, type field). If you do specify the protocol, you must do in the form **Type,Protocol**. Available protocols are:

g This protocol is slower and more reliable than e. It is good for transmission over noisy telephone lines.

e This protocol is faster than g, but it assumes error-free transmission.

For reliable local area networks, you should use the e protocol.

Dialers File

The **Dialers** file (/usr/lib/uucp/Dialers) specifies the initial conversation that must take place on a line before it can be made available for transferring data. This conversation is usually a sequence of ASCII strings that is transmitted and expected, and it is often used to dial a phone number using an ASCII dialer (such as the Automatic Dial Modem).

As shown in the above examples, the fifth field in a **Devices** file entry is an index into the **Dialers** file or a special dialer type. Here an attempt is made to match the fifth field in the **Devices** file with the first field of each **Dialers** file entry. In addition, each odd numbered **Devices** field starting with the seventh position is used as an index into the **Dialers** file. If the match succeeds, the **Dialers** entry is interpreted to perform the dialer negotiations. Each entry in the **Dialers** file has the following format:

dialer substitutions expect-send ...

The **dialer** field matches the fifth and additional odd numbered fields in the **Devices** file. The **substitutions** field is a translate string: the first of each pair of characters is mapped to the second character in the pair. This is usually used to translate = and - into whatever the dialer requires for "wait for dialtone" and "pause."

The remaining **expect-send** fields are character strings. Below are some character strings distributed with the Basic Networking Utilities in the **Dialers** file.

```
penril =W-P "" \d > s\p9\c )-W\p\r\ds\p9\c-) y\c : \E\TP > 9\c OK
ventel =&-% "" \r\p\r\c $ <K\T%%\r>\c ONLINE!
hayes  =,-, "" \dAT\r\c OK\r \EATDT\T\r\c CONNECT
rixon  =&-% "" \d\r\r\c $ s9\c )-W\r\ds9\c-) s\c : \T\r\c $ 9\c LINE
vadiac =K-K "" \005\p *-\005\p-*\005\p-* D\p BER? \E\T\e \r\c LINE
develcon "" "" \pr\ps\c est:\007 \E\D\e \007
micom ""    "" \s\c NAME? \D\r\c GO
direct
att2212c        =+-,    "" \r\c :--: atol2=y,T\T\r\c red
att4000 =,-,    "" \033\r\r\c DEM: \033s0401\c \006 \033s0901\c \
        \006 \033s1001\c \006 \033s1102\c \006 \033dT\T\r\c \006
att2224 =+-,    "" \r\c :--: T\T\r\c red
nls     ""      "" NLPS:000:001:1\N\c
```

The meaning of some of the escape characters (those beginning with "\") used in the **Dialers** file are listed below:

\p	pause (approximately ¼ to ½ second)
\d	delay (approximately 2 seconds)
\D	phone number or token without **Dialcodes** translation
\T	phone number or token with **Dialcodes** translation
\K	insert a BREAK
\E	enable echo checking (for slow devices)
\e	disable echo checking
\r	carriage return
\c	no new-line or carriage return
\n	send new-line
\nnn	send octal number.

Additional escape characters that may be used are listed in the section discussing the **Systems** file.

The penril entry in the **Dialers** file is executed as follows. First, the phone number argument is translated, replacing any = with a **W** (wait for dialtone) and replacing any - with a **P** (pause). The handshake given by the remainder of the line works as follows:

""	Wait for nothing. (In other words, proceed to the next thing.)
\d	Delay for 2 seconds.
>	Wait for a **>**.
s\p9\c	Send an **s**, pause for ½ second, send a **9**, send no terminating new-line
)-W\p\r\ds\p9\c-)	Wait for a). If it is not received, process the string between the - characters as follows. Send a **W**, pause, send a carriage-return, delay, send an **s**, pause, send a **9**, without a new-line, and then wait for the).
y\c	Send a **y**.
:	Wait for a **:**.
\E\TP	Enable echo checking. (From this point on, whenever a character is transmitted, it will wait for the character to be received before doing anything else.) Then, send the phone number. The **\T** means take the phone number passed as an argument and apply the **Dialcodes** translation and the modem function translation specified by field 2 of this entry. Then send a **P**.
>	Wait for a **>**.
9\c	Send a **9** without a new-line.
OK	Waiting for the string **OK**.

Systems File

The **Systems** file (**/usr/lib/uucp/Systems**) contains the information needed by the **uucico** daemon to establish a communication link to a remote computer. Each entry in the file represents a computer that can be called by your computer. In addition, the basic networking software can be configured to prevent any computer that does not appear in this file from logging in on your computer (refer to the section "Other Networking Files" in this chapter for a description of the **remote.unknown** file). More than one entry may be present for a particular computer. The additional entries represent alternative communication paths that will be tried in sequential order. The management of this file is supported by the System Administration Menu subcommand **systemmgmt**.

Using the **Sysfiles**, you can define several files to be used as "Systems" files. See the description of the **Sysfiles** file for details. Each entry in the **Systems** file has the following format:

System-Name Time Type Class Phone Login

Each of these fields is defined in the following section.

System-name
This field contains the node name of the remote computer.

Time This field is a string that indicates the day-of-week and time-of-day when the remote computer can be called. The format of the **Time** field is:

daytime[*;retry*]

The day portion may be a list containing some of the following:

Su Mo Tu We Th Fr Sa
for individual days

Wk for any week-day (Mo Tu We Th Fr)

Any for any day

Never for a passive arrangement with the remote computer. If the **Time** field is **Never**, your computer will never initiate a call to the remote computer. The call must be initiated by the remote computer. In other words, your computer is in a passive mode in respect to the remote computer (see discussion of **Permissions** file).

Here is an example:

```
Wk 1700-0800, Sa, Su
```

This example allows calls from 5:00 p.m. to 8:00 am, Monday through Thursday, and calls any time Saturday and Sunday. The example would be an effective way to call only when phone rates are low, if immediate transfer is not critical.

The **time** portion should be a range of times such as 0800-1230. If no **time** portion is specified, any time of day is assumed to be allowed for the call. A time range that spans 0000 is permitted. For example, **0800-0600** means all times are allowed other than times between 6 a.m. and 8 a.m. An optional subfield, **retry**, is available to specify the minimum time (in minutes) before a retry, following a failed attempt. The default wait is 60 minutes. The subfield separator is a semicolon (;). For example, **Any;9** is interpreted as call any time, but wait at least 9 minutes before retrying after a failure occurs.

Type This field contains the device type that should be used to establish the communication link to the remote computer. The keyword used in this field is matched against the first field of **Devices** file entries as shown below:

```
Systems:  eagle Any ACU,g D1200 3251 ogin: nuucp \
            ssword: Oakgrass

Devices:  ACU ttyl1 - D1200 penril
```

You can define the protocol used to contact the system by adding it on to the **Type** field. The example above shows how to attach the protocol **g** to the device type **ACU**. See the information under the Protocols section in the description of the **Devices** file for details.

Class This field is used to indicate the transfer speed of the device used in establishing the communication link. It may contain a letter and speed (for example, C1200, D1200) to differentiate between classes of dialers (refer to the discussion on the **Devices** file, Class field). Some devices can be used at any speed, so the keyword **Any** may be used. This field must match the **Class** field in the associated **Devices** file entry as shown below:

```
Systems:  eagle Any ACU D1200 NY3251 ogin: nuucp \
            ssword: Oakgrass

Devices:  ACU ttyl1 - D1200 penril
```

If information is not required for this field, use a - as a place holder for the field.

Phone This field is used to provide the phone number (token) of the remote computer for automatic dialers (LAN switches). The phone number is made up of an optional alphabetic abbreviation and a numeric part. If an abbreviation is used, it must be one that is listed in the **Dialcodes** file. For example:

Systems: `eagle Any ACU` **D1200** `NY3251 ogin: nuucp \`
`ssword: Oakgrass`

Dialcodes: `NY 9=1212555`

In this string, an equal sign (=) tells the ACU to wait for a secondary dial tone before dialing the remaining digits. A dash in the string (-) instructs the ACU to pause 4 seconds before dialing the next digit.

If your computer is connected to a LAN switch, you may access other computers that are connected to that switch. The **Systems** file entries for these computers will not have a phone number in the **Phone** field. Instead, this field will contain the token that must be passed on to the switch so it will know which computer your computer wishes to communicate with. (This is usually just the system name.) The associated **Devices** file entry should have a **\D** at the end of the entry to ensure that this field is not translated using the **Dialcodes** file.

Login This field contains login information given as a series of fields and subfields of the format:

expect send

where **expect** is the string that is received and **send** is the string that is sent when the **expect** string is received.

The **expect** field may be made up of subfields of the form:

expect[-*send-expect*]...

where the **send** is sent if the prior **expect** is not successfully read and the **expect** following the **send** is the next expected string. For example, with **login--login**, UUCP will expect **login**. If UUCP gets **login,** it will go on to the next field. If it does not get **login**, it will send nothing followed by a new line, then look for **login** again. If no characters are initially expected from the remote computer, the characters "" (null string) should be used in the first **expect** field. Note that all **send** fields will be sent followed by a new-line unless the **send** string is terminated with a **\c**.

Here is an example of a **Systems** file entry that uses an expect-send string:

```
owl Any ACU 1200 Chicago6013 "" \r ogin:-BREAK-ogin: \
uucpx word: xyzzy
```

This example says send a carriage return and wait for **ogin:** (for `Login:`). If you don't get `ogin`, send a **BREAK**. When you do get `ogin:` send the login name **uucpx**, then when you get **word:** (for Password:), send the password **xyzzy**.

There are several escape characters that cause specific actions when they are a part of a string sent during the login sequence. The following escape characters are useful in UUCP communications:

`\N`	Send or expect a null character (ASCII NUL).
`\b`	Send or expect a backspace character.
`\c`	If at the end of a string, suppress the new-line that is normally sent. Ignored otherwise.
`\d`	Delay two seconds before sending or reading more characters.
`\p`	Pause for approximately ¼ to ½ second.
`\E`	Start echo checking. (From this point on, whenever a character is transmitted, it will wait for the character to be received before doing anything else.)
`\e`	Echo check off.
`\n`	Send a new-line character.
`\r`	Send or expect a carriage-return.
`\s`	Send or expect a space character.
`\t`	Send or expect a tab character.
`\\`	Send or expect a \ character.
`EOT`	Send or expect EOT new-line twice.
`BREAK`	Send or expect a break character.
`\K`	Same as BREAK.
`\ddd`	Collapse the octal digits (ddd) into a single character.

Dialcodes File

The **Dialcodes** file (**/usr/lib/uucp/Dialcodes**) contains the dial-code abbreviations that can be used in the **Phone** field of the **Systems** file. Each entry has the format:

abb dial-seq

where **abb** is the abbreviation used in the **Systems** file **Phone** field and **dial-seq** is the dial sequence that is passed to the dialer when that particular **Systems** file entry is accessed.

The entry

```
jt 9=847-
```

would be set up to work with a **Phone** field in the **Systems** file such as `jt7867`. When the entry containing `jt7867` is encountered, the sequence 9=847-7867 would be sent to the dialer if the token in the dialer-token-pair is `\T`.

Permissions File

The **Permissions** file (**/usr/lib/uucp/Permissions**) specifies the permissions that remote computers have with respect to login, file access, and command execution. There are options that restrict the remote computer's ability to request files and its ability to receive files queued by the local site. Another option is available that specifies the commands that a remote site can execute on the local computer.

How Entries are Structured

Each entry is a logical line with physical lines terminated by a \ to indicate continuation. Entries are made up of options delimited by white space. Each option is a name/value pair in the following format:

name=value

Note that no white space is allowed within an option assignment.

Comment lines begin with a "#" and they occupy the entire line up to a newline character. Blank lines are ignored (even within multi-line entries).

There are two types of **Permissions** file entries:

LOGNAME Specifies the permissions that take effect when a remote computer logs in on (calls) your computer.

MACHINE Specifies permissions that take effect when your computer logs in on (calls) a remote computer.

LOGNAME entries will contain a LOGNAME option and MACHINE entries will contain a MACHINE option.

Considerations

The following items should be considered when using the **Permissions** file to restrict the level of access granted to remote computers:

- All login IDs used by remote computers to login for UUCP communications must appear in one and only one LOGNAME entry.
- Any site that is called whose name does not appear in a MACHINE entry, will have the following default permissions/restrictions:
  - Local send and receive requests will be executed.
  - The remote computer can send files to your computer's /usr/spool/uucppublic directory.
  - The commands sent by the remote computer for execution on your computer must be one of the default commands; usually **rmail**.

Options

This section describes each option, specifies how they are used, and lists their default values.

REQUEST When a remote computer calls your computer and requests to receive a file, this request can be granted or denied. The REQUEST option specifies whether the remote computer can request to set up file transfers from your computer. The string

```
REQUEST=yes
```

specifies that the remote computer can request to transfer files from your computer. The string

```
REQUEST=no
```

specifies that the remote computer cannot request to receive files from your computer. This is the default value. It will be used if the REQUEST option is not specified. The REQUEST option can appear in either a LOGNAME (remote calls you) entry or a MACHINE (you call remote) entry. A note on

security: When a remote machine calls you, unless you have a unique login and password for that machine you don't know if the machine is who it says it is.

SENDFILES When a remote computer calls your computer and completes its work, it may attempt to take work your computer has queued for it. The SENDFILES option specifies whether your computer can send the work queued for the remote computer.

The string

```
SENDFILES=yes
```

specifies that your computer may send the work that is queued for the remote computer as long as it logged in as one of the names in the LOGNAME option. This string is mandatory if your computer is in a "passive mode" with respect to the remote computer.

The string

```
SENDFILES=call
```

specifies that files queued in your computer will be sent only when your computer calls the remote computer. The call value is the default for the SENDFILE option. This option is only significant in LOGNAME entries since MACHINE entries apply when calls are made out to remote computers. If the option is used with a MACHINE entry, it will be ignored.

READ and WRITE

These options specify the various parts of the file system that **uucico** can read from or write to. The READ and WRITE options can be used with either MACHINE or LOGNAME entries.

The default for both the READ and WRITE options is the uucppublic directory as shown in the following strings:

```
READ=/usr/spool/uucppublic
WRITE=/usr/spool/uucppublic
```

The strings

```
READ=/ WRITE=/
```

specify permission to access any file that can be accessed by a local user with "other" permissions.

The value of these entries is a colon separated list of pathnames. The READ option is for requesting files, and the WRITE option for depositing files. One of the values must be the prefix of any full pathname of a file coming in or going out. To grant permission to deposit files in /usr/news as well as the public directory, the following values would be used with the WRITE option:

```
WRITE=/usr/spool/uucppublic:/usr/news
```

It should be pointed out that if the READ and WRITE options are used, all pathnames must be specified because the pathnames are not added to the default list. For instance, if the **/usr/news** pathname was the only one specified in a WRITE option, permission to deposit files in the public directory would be denied.

You should be careful what directories you make accessible for reading and writing by remote systems. For example, you probably wouldn't want remote computers to be able to write over your **/etc/passwd** file so **/etc** shouldn't be open to writes.

NOREAD and NOWRITE

The NOREAD and NOWRITE options specify exceptions to the READ and WRITE options or defaults. The strings

```
READ=/ NOREAD=/etc WRITE=/usr/spool/uucppublic
```

would permit reading any file except those in the /etc directory (and its subdirectories—remember, these are prefixes) and writing only to the default /usr/spool/uucppublic directory. NOWRITE works in the same manner as the NOREAD option. The NOREAD and NOWRITE can be used in both LOGNAME and MACHINE entries.

CALLBACK The CALLBACK option is used in LOGNAME entries to specify that no transaction will take place until the calling system is called back. There are two examples of when you would use CALLBACK. From a security standpoint, if you call back a machine you can be sure it is the machine it says it is. If you are doing long data transmissions, you can choose the machine that will be billed for the longer call.

The string

```
CALLBACK=yes
```

specifies that your computer must call the remote computer back before any file transfers will take place.

The default for the COMMAND option is

```
CALLBACK=no
```

The CALLBACK option is very rarely used. Note that if two sites have this option set for each other, a conversation will never get started.

COMMANDS The COMMANDS option can be hazardous to the security of your system. Use it with extreme care.

The **uux** program will generate remote execution requests and queue them to be transferred to the remote computer. Files and a command are sent to the target computer for remote execution. The COMMANDS option can be used in MACHINE entries to specify the commands that a remote computer can execute on your computer. Note that COMMANDS is not used in a LOGNAME entry; COMMANDS in MACHINE entries define command permissions whether we call the remote system or it calls us.

The string

```
COMMANDS=rmail
```

indicates the default commands that a remote computer can execute on your computer. If a command string is used in a MACHINE entry, the default commands are overridden. For instance, the entry

```
MACHINE=owl:raven:hawk:dove \
COMMANDS=rmail:rnews:lp
```

overrides the COMMAND default so that the computers owl, raven, hawk, and dove can now execute **rmail**, **rnews**, and **lp** on your computer.

In addition to the names as specified above, there can be full pathnames of commands. For example,

```
COMMANDS=rmail:/usr/lbin/rnews:/usr/local/lp
```

specifies that command **rmail** uses the default path. The default paths for your computer are **/bin**, **/usr/bin**, and **/usr/lbin**. When the remote computer specifies **rnews** or **/usr/lbin/rnews** for the command to be executed, **/usr/lbin/rnews** will be executed regardless of the default path. Likewise, **/usr/local/lp** is the **lp** command that will be executed.

Including the ALL value in the list means that any command from the remote computer(s) specified in the entry will be executed. If you use this value, you give the remote computer full access to your computer. BE CAREFUL. This allows far more access than normal users have.

The string

```
COMMANDS=/usr/lbin/rnews:ALL:/usr/local/lp
```

illustrates two points: The ALL value can appear anywhere in the string, and the pathnames specified for **rnews** and **lp** will be used (instead of the default) if the requested command does not contain the full path names for **rnews** or **lp**.

The VALIDATE option should be used with the COMMANDS option whenever potentially dangerous commands like **cat** and **uucp** are specified with the COMMANDS option. Any command that reads or writes files is potentially dangerous to local security when executed by the UUCP remote execution daemon (**uuxqt**).

VALIDATE The VALIDATE option is used in conjunction with the COMMANDS option when specifying commands that are potentially dangerous to your computer's security. It is used to provide a certain degree of verification of the caller's identity. The use of the VALIDATE option requires that privileged computers have a unique login/password for UUCP transactions. An important aspect of this validation is that the login/password associated with this entry be protected. If an outsider gets that information, that particular VALIDATE

option can no longer be considered secure. (VALIDATE is merely an added level of security on top of the COMMANDS option, though it is a more secure way to open command access than ALL.)

Careful consideration should be given to providing a remote computer with a privileged login and password for UUCP transactions. Giving a remote computer a special login and password with file access and remote execution capability is like giving anyone on that computer a normal login and password on your computer. Therefore, if you cannot trust someone on the remote computer, do not provide that computer with a privileged login and password.

LOGNAME The LOGNAME entry

```
LOGNAME=uucpfriend VALIDATE=eagle:owl:hawk
```

specifies that if one of the remote computers that claims to be eagle, owl, or hawk logs in on your computer, it must have used the login **uucpfriend**. As can be seen, if an outsider gets the **uucpfriend** login/password, masquerading is trivial.

But what does this have to do with the COMMANDS option, which only appears in MACHINE entries? It links the MACHINE entry (and COMMANDS option) with a LOGNAME entry associated with a privileged login. This link is needed because the execution daemon is not running while the remote computer is logged in. In fact, it is an asynchronous process with no knowledge of what computer sent the execution request. Therefore, the real question is how does your computer know where the execution files came from?

Each remote computer has its own "spool" directory on your computer. These spool directories have write permission given only to the UUCP programs. The execution files from the remote computer are put in its spool directory after being transferred to your computer. When the **uuxqt** daemon runs, it can use the spool directory name to find the MACHINE entry in the **Permissions** file and get the COMMANDS list, or if the computer name does not appear in the **Permissions** file, the default list will be used.

The following example shows the relationship between the MACHINE and LOGNAME entries:

```
MACHINE=eagle:owl:hawk REQUEST=yes \
COMMANDS=rmail:/usr/lbin/rnews \
READ=/  WRITE=/

LOGNAME=uucpz VALIDATE=eagle:owl:hawk \
REQUEST=yes SENDFILES=yes \
READ=/  WRITE=/
```

The value in the COMMANDS option means that remote mail and **/usr/lbin/rnews** can be executed by remote users.

In the first entry, you must make the assumption that when you want to call one of the computers listed, you are really calling either **eagle**, **owl**, or **hawk**. Therefore, any files put into one of the **eagle**, **owl**, or **hawk** spool directories is put there by one of those computers. If a remote computer logs in and says that it is one of these three computers, its execution files will also be put in the privileged spool directory. You therefore have to validate that the computer has the privileged login **uucpz**.

You may want to specify different option values for the computers your computer calls that are not mentioned in specific MACHINE entries. This may occur when there are many computers calling in, and the command set changes from time to time. The name "OTHER" for the computer name is used for this entry as shown below:

```
MACHINE=OTHER \
COMMANDS=rmail:rnews:/usr/lbin/Photo:/usr/lbin/xp
```

All other options available for the MACHINE entry may also be set for the computers that are not mentioned in other MACHINE entries.

Combining MACHINE and LOGNAME Entries

It is possible to combine MACHINE and LOGNAME entries into a single entry where the common options are the same. For example, the two entries

```
MACHINE=eagle:owl:hawk REQUEST=yes \
  READ=/  WRITE=/

LOGNAME=uucpz REQUEST=yes SENDFILES=yes \
  READ=/  WRITE=/
```

share the same REQUEST, READ, and WRITE options. These two entries can be merged as shown below:

```
MACHINE=eagle:owl:hawk REQUEST=yes \
LOGNAME=uucpz SENDFILES=yes \
  READ=/  WRITE=/
```

Poll File

The **Poll** file (/usr/lib/uucp/Poll) contains information for polling remote computers. Each entry in the **Poll** file contains the name of a remote computer to call, followed by a <tab> character (a space won't work), and finally the hours the computer should be called. The format of entries in the **Poll** file are:

sys-name hour ...

For example the entry:

```
eagle  0 4 8 12 16 20
```

will provide polling of computer **eagle** every four hours.

The **uudemon.poll** script does not actually perform the poll. It merely sets up a polling work file (always named C.**file**), in the spool directory that will be seen by the scheduler, which is started by **uudemon.hour**.

Sysfiles File

The /usr/lib/uucp/Sysfiles file lets you assign different files to be used by **uucp** and **cu** as **Systems**, **Devices**, and **Dialers** files. Here are some cases where this optional file may be useful.

- You may want different **Systems** files so requests for login services can be made to different addresses than **uucp** services.
- You may want different **Dialers** files to use different handshaking for **cu** and **uucp**.
- You may want to have multiple **Systems**, **Dialers**, and **Devices** files. The **Systems** file in particular may become large, making it more convenient to split it into several smaller files.

The format of the **Sysfiles** file is

service=*w* **systems=***x*:*x* **dialers=***y*:*y* **devices=***z*:*z*

where *w* is replaced by **uucico**, **cu**, or both separated by a colon; *x* is one or more files to be used as the **Systems** file, with each file name separated by a colon and read in the order presented; *y* is one or more files to be used as the **Dialers** file; and *z* is one or more files to be used as the **Devices** file. Each file is assumed to be relative to the **/usr/lib/uucp** directory, unless a full path is given. A backslash-carriage return (\<**return**>) can be used to continue an entry on to the next line.

Here's an example of using a local **Systems** file in addition to the usual **Systems** file:

```
service=uucico:cu    systems=Systems:Local_Systems
```

If this is in **/usr/lib/uucp/Sysfiles**, then both **uucico** and **cu** will first look in **/usr/lib/uucp/Systems**. If the system they're trying to call doesn't have an entry in that file, or if the entries in the file fail, then they'll look in **/usr/lib/uucp/Local_Systems**.

When different **Systems** files are defined for **uucico** and **cu** services, your machine will store two different lists of Systems. You can print the **uucico** list using the **uuname** command or the **cu** list using the **uuname –c** command.

Other Networking Files

There are three other files that impact the use of basic networking facilities. In most cases, the default values are fine and no changes are needed. If you want to change them, however, use any standard UNIX system text editor (**ed** or **vi**).

Maxuuxqts	This file defines the maximum number of **uuxqt** programs that can run at once.
Maxuuscheds	This file defines the maximum number of **uusched** programs that can run at once.
remote.unknown	This file is a shell script that executes when a machine that is not in any of the **Systems** starts a conversation. It will log the conversation attempt and fail to make a connection. If you change the permissions of this file so it cannot execute (**chmod 000 remote.unknown**), your system will accept any conversation requests.

Administrative Files

The basic networking administrative files are described below. These files are created in spool directories to lock devices, hold temporary data, or keep information about remote transfers or executions.

TM (temporary data file)

These data files are created by Basic Networking processes under the spool directory (i.e., /usr/spool/uucp/X) **when a file is received from another computer. The directory X has the same name as the remote computer that is sending the file. The names of the temporary data files have the format:**

TM.pid.ddd

where *pid* is a process-ID and *ddd* is a sequential three digit number starting at 0.

When the entire file is received, the **TM.***pid.ddd* file is moved to the pathname specified in the **C.***sysnxxxx* file (discussed below) that caused the transmission. If processing is abnormally terminated, the **TM.***pid.ddd* file may remain in the X directory. These files should be automatically removed by **uucleanup**.

LCK (lock file)

Lock files are created in the /usr/spool/locks directory for each device in use. Lock files prevent duplicate conversations and multiple attempts to use the same calling device. The names of lock files have the format:

LCK..str

where **str** is either a device or computer name. These files may remain in the spool directory if the communications link is unexpectedly dropped (usually on computer crashes). The lock files will be ignored (removed) after the parent process is no longer active. The lock file contains the process ID of the process that created the lock.

C. (work file) Work files are created in a spool directory when work (file transfers or remote command executions) has been queued for a remote computer. The names of work files have the format:

C.sysnxxxx

where *sys* is the name of the remote computer, *n* is the ASCII character representing the grade (priority) of the work, and *xxxx* is the four digit job sequence number assigned by UUCP. Work files contain the following information:

- ☐ Full pathname of the file to be sent or requested

  Full pathname of the destination or userle name
- ☐ User login name
- ☐ List of options
- ☐ Name of associated data file in the spool directory. If the **uucp –c** or **uuto –p** option was specified, a dummy name **(D.0)** is used
- ☐ Mode bits of the source file
- ☐ Remote user's login name to be notified upon completion of the transfer

D. (Data file) Data files are created when it is specified in the command line to copy the source file to the spool directory. The names of data files have the following format:

D.*systmxxxxyyy*

where *systm* is the first five characters in the name of the remote computer, *xxxx* is a four-digit job sequence number assigned by **uucp**. The four digit job sequence number may be followed by a sub-sequence number, *yyy* that is used when there are several **D.** files created for a work (C.) file.

X. (Execute file)

Execute files are created in the spool directory prior to remote command executions. The names of execute files have the following format:

X.*sysnxxxx*

where *sys* is the name of the remote computer, *n* is the character representing the grade (priority) of the work, and *xxxx* is a four digit sequence number assigned by UUCP. Execute files contain the following information:

- ☐ Requester's login and computer name.
- ☐ Name of file(s) required for execution.
- ☐ Input to be used as the standard input to the command string.
- ☐ Computer and file name to receive standard output from the command execution.
- ☐ Command string.
- ☐ Option lines for return status requests.

uucp Error Messages

ASSERT Error Messages

When a process is aborted the system records ASSERT error messages in **/usr/spool/uucp/.Admin/errors**. These messages include the file name, **sccsid**, line number, and the text listed below. In most cases, these errors are the result of file system problems. Use the "errno" (when present) to investigate the problem. If "errno" is present in a message, it is shown as () in the following list.

Error Message	Description/Action
CAN'T OPEN	An open() or fopen() failed.
CAN'T WRITE	A write(), fwrite(), fprint(), etc. failed.
CAN'T READ	A read(), fgets(), etc. failed.
CAN'T CREATE	A create() call failed.
CAN'T ALLOCATE	A dynamic allocation failed.
CAN'T LOCK	An attempt to make a LCK (lock) file failed. In some cases, this is a fatal error.
CAN'T STAT	A stat() call failed.
CAN'T CHMOD	A chmod() call failed.
CAN'T LINK	A link() call failed.
CAN'T CHDIR	A chdir() call failed.
CAN'T UNLINK	An unlink() call failed.
WRONG ROLE	This is an internal logic problem.
CAN'T MOVE TO CORRUPTDIR	An attempt to move some bad C. or X. files to the /usr/spool/uucp/.Corrupt directory failed. The directory is probably missing or has wrong modes or owner.

Error Message	Description/Action
CAN'T CLOSE	A close() or fclose() call failed.
FILE EXISTS	The creation of a C. or D. file is attempted, but the file exists. This occurs when there is a problem with the sequence file access. This usually indicates a software error.
No uucp server	A TCP/IP call is attempted, but there is no server for UUCP.
BAD UID	The uid cannot be found in the **/etc/passwd** file. The file system is in trouble, or the **/etc/passwd** file is inconsistent.
BAD LOGIN_UID	The uid cannot be found in the **etc/passwd** file. The file system is in trouble, or the **/etc/passwd** file is inconsistent.
ULIMIT TOO SMALL	The ulimit for the current user process is too small. File transfers may fail, so transfer is not attempted.
BAD LINE	There is a bad line in the **Devices** file; there are not enough arguments on one or more lines.
FSTAT FAILED IN EWRDATA	There is something wrong with the ethernet media.
SYSLST OVERFLOW	An internal table in gename.c overflowed. A big/strange request was attempted. Contact your service representative.
TOO MANY SAVED C FILES	An internal table in gename.c overflowed. A big/strange request was attempted. Contact your service representative.
RETURN FROM fixline ioctl	An **ioctl**, which should never fail, failed. There is a system driver problem.

Error Message	Description/Action
BAD SPEED	A bad line speed appears in the **Devices/Systems** files (Class field).
PERMISSIONS file: BAD OPTION	There is a bad line or option in the **Permissions** file. Fix it immediately!
PKCGET READ	The remote machine probably hung up. No action is required.
PKXSTART	The remote machine aborted in a non-recoverable way. This can generally be ignored.
SYSTAT OPEN FAIL	There is a problem with the modes of **/usr/lib/uucp/.Status**, or there is a file with bad modes in the directory.
TOO MANY LOCKS	There is an internal problem!
XMV ERROR	There is a problem with some file or directory. It is likely the spool directory, since the modes of the destinations were suppose to be checked before this process was attempted.
CAN'T FORK	An attempt to fork and exec failed. The current job should not be lost, but will be attempted later (**uuxqt**). No action need be taken.

STATUS Error Messages

Status error messages are messages that are stored in the /usr/spool/uucp/.Status directory. This directory contains a separate file for each remote machine that your workstation attempts to communicate with. These individual machine files contain status information on the attempted communication, whether it was successful or not. What follows is a list of the most common error messages that may appear in these files.

Error Message	Description/Action
OK	Things are OK.
NO DEVICES AVAILABLE	There is currently no device available for the call. Check to see that there is a valid device in the **Devices** file for the particular system. Check the **Systems** file for the device to be used to call the system.
WRONG TIME TO CALL	A call was placed to the system at a time other than what is specified in the **Systems** file.
TALKING	Self explanatory.
LOGIN FAILED	The login for the given machine failed. It could be a wrong login/password, wrong number, a very slow machine, or failure in getting through the Dialer-Token-Pairs script.
CONVERSATION FAILED	The conversation failed after successful startup. This usually means that one side went down, the program aborted, or the line (link) was dropped.
DIAL FAILED	The remote machine never answered. It could be a bad dialer or the wrong phone number.
BAD LOGIN/MACHINE COMBINATION	The machine called us with a login/machine name that does not agree with the **Permissions** file. This could be an attempt to masquerade!

Error Message	Description/Action
DEVICE LOCKED	The calling device to be used is currently locked and in use by another process.
ASSERT ERROR	An ASSERT error occurred. Check the **/usr/spool/uucp/.Admin/errors** file for the error message and refer to the section ASSERT Error Messages.
SYSTEM NOT IN Systems	The system is not in the **Systems** file.
CAN'T ACCESS DEVICE	The device tried does not exist or the modes are wrong. Check the appropriate entries in the **Systems** and **Devices** files.
DEVICE FAILED	The open of the device failed.
WRONG MACHINE NAME	The called machine is reporting a different name than expected.
CALLBACK REQUIRED	The called machine requires that it calls your 3B2 Computer.
REMOTE HAS A LCK FILE FOR ME	The remote site has a LCK file for your 3B2 Computer. They could be trying to call your machine. If they have an older version of Basic Networking, the process that was talking to your machine may have failed leaving the LCK file. If they have the new version of Basic Networking, and they are not communicating with your 3B2 Computer, then the process that has a LCK file is hung.
REMOTE DOES NOT KNOW ME	The remote machine does not have the node name of your 3B2 Computer in its Systems file.
REMOTE REJECT AFTER LOGIN	The login used by your 3B2 Computer to login does not agree with what the remote machine was expecting.
REMOTE REJECT, UNKNOWN MESSAGE	The remote machine rejected the communication with your 3B2 Computer for an unknown reason. The remote machine may not be running a standard version of Basic Networking.

Error Message	Description/Action
STARTUP FAILED	Login succeeded, but initial handshake failed.
CALLER SCRIPT FAILED	This is usually the same as "DIAL FAILED." However, if it occurs often, suspect the caller script in the **dialers** file. Use **uutry** to check.

Administrative Files and Directories

Appendix A describes directories and files of interest to a system administrator.

Directories

The directories of the **root** file system (/) are as follows:

bin	Directory containing public commands.
boot	Directory containing configurable object files created by the **/etc/mkboot**(1M) program.
dev	Directory containing special files that define all of the devices on the system.
etc	Directory containing administrative programs and tables. mount utilities packages for installation and removal (**/install** file system).
lib	Directory containing public libraries.
lost+found	Directory used by **fsck**(1M) to save disconnected files. of the operating system from floppy disks.
tmp	Directory used for temporary files.
usr	Directory used to mount the **/usr** file system.

Files

The following files and directories are important in the administration of the workstation:

- **/etc/fstab**
- **/etc/gettydefs**
- **/etc/group**
- **/etc/init.d Directory**
- **/etc/inittab /etc/motd**
- **/etc/passwd**
- **/etc/profile**
- **/etc/rc0**
- **/etc/rc0.d Directory**
- **/etc/rc2**
- **/etc/rc2.d Directory**
- **/etc/rc.d Directory**
- **/etc/rc3**
- **/etc/rc3.d Directory**
- **/etc/shutdown**
- **/etc/shutdown.d Directory**
- **/etc/TIMEZONE**
- **/etc/utmp**
- **/usr/adm/sulog**
- **/usr/lib/cron/log**
- **/usr/lib/help/HELPLOG**
- **/usr/options Directory.**
- **/usr/spool/cron/crontabs Directory**

Each of these files is briefly described in this appendix.

/etc/fstab

The **/etc/fstab** file is used as an argument to the **/etc/mountall** command. The **fstab** file specifies the file system(s) to be mounted by **/etc/mountall.** The format of the file is the block device name followed by the mount point name. (See the **mountall**(1M) manual page in the *IRIS-4D/ System Administrator's Reference Manual* for additional information.)

```
/dev/root / efs rw,raw=/dev/rroot 0 0
/dev/usr /usr efs rw 0 0
```

Figure A-1: Typical **/etc/fstab** File

/etc/gettydefs

The **/etc/gettydefs** file contains information that is used by **/etc/getty** to set the speed and terminal settings for a line. The **getty** command accesses the **gettydefs** file with a label. The general format of the **gettydefs** file is as follows:

label# initial-flags # final-flags #login-prompt #next-label

Each line entry in the **gettydefs** file is followed by a blank line. (Refer to the **gettydefs**(4) manual page in the *IRIS-4D Programmer's Reference Manual* for complete information.)

/etc/group

The **/etc/group** file describes each group to the system. An entry is added for each new group. Each entry in the file is one line and consists of four fields, which are separated by a colon (:):

group name:password:group id:login names

Explanations for these fields are as follows:

group name	The first field defines the group name. The group name is from three to six characters long. The first character is alphabetic. The rest of the characters are alphanumeric. No uppercase characters appear.
password	The second field contains the encrypted group password. The encrypted group password contains 13 bytes (characters). The actual password is limited to a maximum of 8 bytes. The encrypted password can be followed by a comma and up to 4 more bytes of password aging information. The use of group passwords is discouraged.
group id	The third field contains the group identification number, which must be between 0 and 60,000. Group identification numbers 0 through 99 are reserved; 0 indicates the super-user (root). Commas are not entered in this field.
login names	The fourth field contains a list of all login names in the group. Names in the list are separated by commas. The names listed may use the **/etc/newgrp** command to become a member of the group.

Figure A-2 shows a typical **/etc/group** file.

```
root::0:root
other::1:
bin::2:root,bin,daemon
sys::3:root,bin,sys,adm
adm::4:root,adm,daemon
mail::6:root
rje::8:rje,shqer
daemon::12:root,daemon
```

Figure A-2: Typical **/etc/group** File

/etc/init.d Directory

The **/etc/init.d** directory contains executable files used in upward and downward transitions to all system run levels. These files are linked to files beginning with **S** (start) or **K** (stop) in **/etc/rc*n*.d**, where *n* is the appropriate run level. Files are not executed from this directory. They are only executed from **/etc/rc*n*.d** directories.

/etc/inittab

The **/etc/inittab** file contains instructions for the **/etc/init** command. The instructions define the processes that are to be created or terminated for each initialization state. Initialization states are called run levels or run-states. By convention, run level 1 (or S or s) is single-user mode; run level 2 is multi-user mode. Chapter 3, "Processor Operations," summarizes the various run levels and describes their uses. (See the **inittab**(4) manual page in the *IRIS-4D Programmer's Reference Manual* for additional information.) The typical entry is a series of fields separated by a colon (:):

identification:run-state:action:process

Explanations for these fields are as follows:

identification	The identification field is a one- or two-character identifier for the line entry. The identifier is unique for a line.
run-state	The run-state defines the run level in which the entry is to be processed.
action	The action field defines how **/etc/init** treats the process field. (Refer to the **inittab**(4) manual page in the *IRIS-4D Programmer's Reference Manual* for complete information.)
process	The process field defines the shell command that is to be executed.

/etc/master.d Directory

The **/etc/master.d** directory contains files that define the configuration of hardware devices, software drivers, system parameters and aliases. The files are used by **/etc/lboot** to obtain device information for the generation of device driver and configurable module files. The first step in reconfiguring the system to run with different tunable parameters is to edit the appropriate files in the **/etc/master.d** directory. (Refer to the **/etc/master**(4) manual page in the *IRIS-4D Programmer's Reference Manual* for additional information.)

/etc/motd

The **/etc/motd** file contains the message-of-the-day. The message-of-the-day is output by instructions in the **/etc/profile** file after a successful login. This message should be kept short and to the point. The **/usr/news** file(s) should be used for lengthy, more explicit messages.

/etc/passwd

The **/etc/passwd** file identifies each user to the system. An entry is added for each new user. Each entry in the file is one line and consists of seven fields. The fields are separated by a colon (:):

login name:passwd:user:group:account:login directory:program

Explanations for these fields are as follows:

login name	The first field defines the login name. The login name is from three to six characters long. The first character is alphabetic. The rest of the characters are alphanumeric. No uppercase characters appear.
passwd	The second field contains the encrypted login password. The encrypted login password contains 13 bytes (characters). The actual password is limited to a maximum of 8 bytes. The encrypted password can be followed by a comma and up to 4 more bytes of password aging information.
user id	The third field contains the user identification number, which must be between 0 and 60,000. Group identification numbers 0 through 99 are reserved; 0 indicates the super-user (root). Commas are not entered in this field.
group id	The fourth field contains the group identification number, which must be between 0 and 60,000. Group identification numbers 0 through 99 are reserved; 0 indicates the super-user (root). Commas are not entered in this field.
account	The fifth field is used by accounting programs. This field typically contains the user name, department number, and bin number.
login directory	The sixth field defines the full path name of the login directory.
program	The seventh field defines the program to be executed after login. If it is null, the shell (**/bin/csh**) is invoked.

/etc/profile

The default profile for all users is in the **/etc/profile** file. The standard (default) environment for all Bourne shell users is established by the instructions in the **/etc/profile** file. The system administrator can modify this file to set options for the **root** login. For example, the following can be added to the **/etc/profile** for the **root** login to cause the erase character to back up and to set the TERM variable.

```
if [ ${LOGNAME} = root ]
        then
                stty echoe
                echo "Enter TERM: \c"
                read TERM
                export TERM
```

Figure A-3 shows the default profile.

```
# The profile that all logins get before using their own .profile.
trap "" 2 3
export LOGNAME
. /etc/TIMEZONE
# Login and -su shells get /etc/profile services.
# -rsh is given its environment in its .profile.
case "$0" in
-su )
                        export PATH
                        ;;
-sh )
                        export PATH
        # Allow the user to break the Message-Of-The-Day only.
                        trap "trap '' 2" 2
                        cat -s /etc/motd
                        trap "" 2
        if mail -e
                        then
                        echo "you have mail"
                        fi
        if [ ${LOGNAME} != root ]
                        then
                        news -n
                        fi
                        ;;
esac
umask 022
trap 2 3
```

Figure A-3: Standard **/etc/profile** File

/etc/rc0

The **/etc/rc0** file contains a shell script that is executed by **/etc/shutdown** for transitions to single-user state, and by **/etc/init** on transitions to run level 0. Files in the **/etc/shutdown.d** and **/etc/rc0.d** directories are executed when **/etc/rc0** is run. The file **K00ANNOUNCE** in **/etc/rc0.d** prints the message "`System services are now being stopped.`" Any task that you want executed when the system is taken to run level 0 can be done by adding a file to the **/etc/shutdown.d** directory.

Figure A-4 shows a typical **/etc/rc0** file.

```
# "Run Commands" for init state 0
# Leaves the system in a state where it is safe to turn off the power
# or go to firmware.
stty sane tab3 2>/dev/null
echo 'The system is coming down.  Please wait.'
if [ -d /etc/shutdown.d ]
then
                        for f in /etc/shutdown.d/*
                        { if [ -s $f ]
                        then
                        /bin/sh ${f}
                        fi
                        }
fi
#                       End of historical section
if [ -d /etc/rc0.d ]
then
                        for f in /etc/rc0.d/K*
                        {
                        if [ -s ${f} ]
                        then
                        /bin/sh ${f} stop
                        fi
                        }
 .
 .
 .
```

Figure A-4: Typical **/etc/rc0** File (Sheet 1 of 2)

```
.
.
.
# system cleanup functions ONLY (things that end fast!)
for f in /etc/rc0.d/S*
                        {
                        if [ -s ${f} ]
                        then
                        /bin/sh ${f} start
                        fi
                        }
fi
trap "" 15
kill -15 -1
sleep 10
/etc/killall  9
sleep 10
sync;sync;sync
/etc/umountall
stty sane 2>/dev/null
sync;  sync
echo '
The system is down.'
sync
```

Figure A-4: Typical **/etc/rc0** File (Sheet 2 of 2)

/etc/rc0.d Directory

The **/etc/rc0.d** directory contains files executed by **/etc/rc0** for transitions to system run levels 0, 5, and 6. Files in this directory are linked from the **/etc/init.d** directory, and begin with either a **K** or an **S**. **K** indicates processes that are stopped, and **S** indicates processes that are started when entering run level 0.

/etc/rc2

The **/etc/rc2** file contains a shell script that is executed by **/etc/init** on transitions to run level 2 (multi-user state). Executable files in the **/etc/rc.d** and any executable files beginning with **S** or **K** in **/etc/rc2.d** directories are executed when **/etc/rc2** is run. All files in **rc2.d** are linked from files in the **/etc/init.d** directory. The following are descriptions of some of those files. These files are prefixed with an **S** or a **K** and a number in the **/etc/rc2.d** directory.

MOUNTFILESYS	Sets up and mounts file systems. Builds the mount table and mounts the **root** (/) and user (**/usr**) file systems. Makes the **/usr/tmp** directory, cleaning up (deleting) any previous files in that directory.
autoconfig	Makes a **/unix** if self-configuration occurred during the boot sequence. The new in-memory operating system is copied to **/unix**.
cron	Starts the **cron** daemon by executing **/etc/cron**.
sysetup	Removes the **/etc/ps_data** to force the **/bin/ps** command to read the **/unix** file. Outputs the system configuration if the **/etc/prtconf** command exists. Outputs the system trademark information.
uucp	When basic networking is added to the system, the **uucp** file is added to this directory. The **uucp** file deletes **uucp** locks (LCK*), status files (STST*), and temporary files (TM*) under the **/usr/spool/uucp** directory structure.
lp	When line printer spooling is added to the system, the **lp** file is added to the **rc.d**. The **lp** file removes the spooler lock file and starts the scheduler.

Other files may also be added to **/etc/rc2.d** and **/etc/rc.d** directories as a function of adding hardware or software to the system. Figure A-5 shows a typical **/etc/rc2** file.

```
# "Run Commands" executed when the system is changing to init state 2,
# traditionally called "multi-user".
. /etc/TIMEZONE
# Pickup start-up packages for mounts, daemons, services, etc.
set `who -r`
if [ $9 = "S" ]
then
   echo 'The system is coming up.  Please wait.'
   BOOT=yes
   if [ -f /etc/rc.d/PRESERVE ]# historical segment for vi and ex
                           then
                           mv /etc/rc.d/PRESERVE /etc/init.d
                           ln /etc/init.d/PRESERVE /etc/rc2.d/S02PRESERVE
                           fi
elif [ $7 = "2" ]
then
                           echo 'Changing to state 2.'
                           if [ -d /etc/rc2.d ]
                           then
                           for f in /etc/rc2.d/K*
                           {
                           if [ -s ${f} ]
                           then
                           /bin/sh ${f} stop
                           fi
                           }
                           fi
fi
 .
 .
 .
```

Figure A-5: Typical /etc/rc2 File (Sheet 1 of 2)

```
.
.
.
if [ -d /etc/rc2.d ]
then
                        for f in /etc/rc2.d/S*
                        {
                        if [ -s ${f} ]
                        then
                        /bin/sh ${f} start
                        fi
                        }
fi
if [ "${BOOT}" = "yes" ]
then
                        stty sane tab3 2>/dev/null
fi
if [ "${BOOT}" = "yes" -a -d /etc/rc.d ]
then
                        for f in `ls /etc/rc.d`
                        {
                        if [ ! -s /etc/init.d/${f} ]
                        then
                        /bin/sh /etc/rc.d/${f}
                        fi
                        }
fi
if [ "${BOOT}" = "yes" -a $7 = "2" ]
then
                        echo 'The system is ready.'
elif [ $7 = "2" ]
then
                        echo 'Change to state 2 has been completed.'
fi
```

Figure A-5: Typical **/etc/rc2** File (Sheet 2 of 2)

/etc/rc2.d Directory

The /etc/rc2.d directory contains files executed by /etc/rc2 for transitions to system run level 3. Files in this directory are linked from the /etc/init.d directory, and begin with either a K or an S. K indicates processes that should be stopped, and S indicates processes that should be started when entering run level 2.

/etc/rc.d Directory

The /etc/rc.d directory contains executable files that do the various functions needed to initialize the system to run level 2. The files are executed when /etc/rc2 is run. (Files contained in this directory prior to UNIX System release 3.0 were moved to /etc/rc2.d. This directory is only maintained for compatibility reasons.)

/etc/shutdown

The /etc/shutdown file contains a shell script to shut down the system gracefully in preparation for system backup or scheduled downtime. After stopping all nonessential processes, the shutdown script executes files in the /etc/shutdown.d directory by calling /etc/rc0 for transitions to run level s or S. For transitions to other run levels, the shutdown script calls /etc/init.

.
.
.

/etc/TIMEZONE

The /etc/TIMEZONE file sets the time zone shell variable TZ. The TZ variable is initially established for the system via the System Administration setup function. The TZ variable in the TIMEZONE file is changed by the System Administration timezone command (sysadm timezone). The TZ variable can be redefined on a user (login) basis by setting the variable in the associated .profile. The TIMEZONE file is executed by /etc/rc2. Figure A-6 shows a typical /etc/TIMEZONE file.

The format of the **TZ** is as follows:

TZ=*TTT#SSS*

Explanations for the fields of the **TZ** variable are as follows:

TTT	The three-character abbreviation for the local time zone.
#	The number of hours that the local time zone differs from Greenwich Mean Time (GMT). This field can be entered as a positive or negative number.
SSS	The three-character abbreviation for the local daylight savings time zone. This field is entered only if daylight savings time is observed.

```
# Set timezone environment to default for the machine
TZ=PST8PDT
export TZ
```

Figure A-6: Typical **/etc/TIMEZONE** File

/etc/utmp

The **/etc/utmp** file contains information on the run-state of the system. This information is accessed with a **who -a** command.

/etc/wtmp

The **/etc/wtmp** file contains a history of system logins. The owner and group of this file must be **adm**, and the access permissions must be 664. Each time **login** is run this file is updated. As the system is accessed, this file increases in size. Periodically, this file should be cleared or truncated. The command line **>/etc/wtmp** when executed by **root** creates the file with nothing in it. The following command line limits the size of the **/etc/wtmp** file to the last 3600 characters in the file:

```
tail -3600c /etc/wtmp > /tmp/wtmp; mv /tmp/wtmp /etc/wtmp
```

Note that **/etc/cron**, **/etc/rc0**, or **/etc/rc2** can be used to clean up the **wtmp** file. To use one of these functions, add the appropriate command line to the **/usr/spool/cron/crontab/root**, **/etc/shutdown.d/...**, or **/etc/rc.d/**, **rc2.d**, **rc3.d ...** file.

/usr/adm/sulog

The **/usr/adm/sulog** file contains a history of substitute user (**su**) command usage. As a security measure, this file should not be readable by **others**. The **/usr/adm/sulog** file should be periodically truncated to keep the size of the file within a reasonable limit. Note that **/etc/cron**, **/etc/rc0**, or **/etc/rc2** can be used to clean up the **sulog** file. To use one of these functions, add the appropriate command line to the **/usr/spool/cron/crontab/root**, **/etc/shutdown.d/...**, or **/etc/rc.d/**, **rc2.d**, **rc3.d ...** file. The following command lines limit the size of the log file to the last 100 lines in the file:

```
tail -100 /usr/adm/sulog > /tmp/sulog
mv /tmp/sulog /usr/adm/sulog
```

Figure A-7 shows the contents of a typical **/usr/adm/sulog** file.

```
SU 08/18 12:35 + console root-sysadm
SU 08/18 16:11 + console root-sysadm
SU 08/18 16:16 + console root-sysadm
SU 08/18 23:45 + tty?? root-uucp
SU 08/19 11:53 + console root-sysadm
SU 08/19 15:25 + console root-sysadm
SU 08/19 23:45 + tty?? root-uucp
SU 08/20 10:16 + console root-adm
SU 08/20 10:33 + tty24 rar-root
SU 08/20 10:42 + console root-sysadm
SU 08/20 10:59 + console root-root
SU 08/20 11:01 + console root-sysadm
SU 08/20 12:36 + tty11 bin-bin
SU 08/20 12:37 + tty11 tws-bin
SU 08/20 14:42 - tty24 awa-sys
SU 08/20 14:47 - tty24 awa-sys
SU 08/20 14:48 + tty24 awa-root
SU 08/20 15:44 + console root-sysadm
```

Figure A-7: Typical **/usr/adm/sulog** File

/usr/lib/cron/log

A history of all actions taken by **/etc/cron** is recorded in the **/usr/lib/cron/log** file. The **/usr/lib/cron/log** file should be periodically truncated to keep the size of the file within a reasonable limit. Note that **/etc/cron**, **/etc/rc0**, or **/etc/rc2** can be used to clean up the **/usr/lib/cron/log** file. To use one of these functions to limit the size of a log file, add the appropriate command line to the **/usr/spool/cron/crontab/root**, **/etc/shutdown.d/...**, or **/etc/rc.d/ rc2.d, rc3.d ...** file, as applicable. The following command line limits the size of the log file to the last 100 lines in the file:

tail -100 /usr/lib/cron/log > /tmp/log; mv /tmp/log /usr/lib/cron/log

Figure A-8 shows the information typically found in the **/usr/lib/cron/log** file.

```
! *** cron started ***   pid = 237 Sun Aug 19 14:06:45 1984
>  CMD: /usr/lib/uucp/uudemon.hour > /dev/null
>  root 251 c Sun Aug 19 14:11:00 1984
<  root 251 c Sun Aug 19 14:11:01 1984
>  CMD: /usr/lib/uucp/uudemon.poll > /dev/null
>  root 370 c Sun Aug 19 14:30:00 1984
<  root 370 c Sun Aug 19 14:30:03 1984
>  CMD: /usr/lib/uucp/uudemon.hour > /dev/null
>  root 417 c Sun Aug 19 14:41:01 1984
<  root 417 c Sun Aug 19 14:41:02 1984
>  CMD: /usr/lib/uucp/uudemon.poll > /dev/null
>  root 452 c Sun Aug 19 15:01:00 1984
<  root 452 c Sun Aug 19 15:01:04 1984
>  CMD: /usr/lib/uucp/uudemon.hour > /dev/null
>  root 460 c Sun Aug 19 15:11:00 1984
<  root 460 c Sun Aug 19 15:11:00 1984
>  CMD: /usr/lib/uucp/uudemon.poll > /dev/null
>  root 541 c Sun Aug 19 15:30:00 1984
<  root 541 c Sun Aug 19 15:30:07 1984
```

Figure A-8: Typical **/usr/lib/cron/log** File

/usr/lib/spell/spellhist

If the Spell Utilities is installed, a history of all words that **spell**(1) fails to match is kept in the **/usr/lib/spell/spellhist** file. Periodically, this file should be reviewed for words that should be added to the dictionary. After the **spellhist** file is reviewed, it can be cleared.

/usr/news

The **/usr/news** directory contains news files. The file names are descriptive of the contents of the files; they are analogous to headlines. When a user reads the news, using the **news** command, an empty file named **.news_time** is created in his or her login directory. The date (time) of this file is used by the **news** command to determine if a user has read the latest news file(s).

/usr/spool/cron/crontabs

The **/usr/spool/cron/crontabs** directory contains crontab files for **adm, root,** and **sys** logins. Providing their lognames are in the **/usr/lib/cron/cron.allow** file, users can establish their own **crontabs** file using the **crontab** command. If the **cron.allow** file does not exist, the **/usr/lib/cron/cron.deny** file is checked to determine if the user is denied the use of the **crontab** command.

As **root,** you can either use the **crontab**(1) command or edit the appropriate file under **/usr/spool/cron/crontabs** to make the desired entries. Revisions to the file take effect at the next reboot. The line entry format of a **/usr/spool/cron/crontabs/***logname* file is as follows:

minute hour day month day-of-week command

The various fields of a **crontabs/***logname* line entry are the following:

minute	The minutes field is a one- or two-digit number in the range 0 through 59.
hour	The hour field is a one- or two-digit number in the range 0 through 24.
day	The day field is the numerical day of the month in the range 1 through 31.
month	The month field is the numerical month of the year in the range 1 through 12.
day-of-week	The day-of-week field is the numerical day of the week where Sunday is 0, Monday is 1, . . . and Saturday is 6.
command	The command field is the program or command that is executed at the time specified by the first five fields.

The following syntax applies to the first five fields:

- Two numbers separated by a minus indicates an inclusive range of numbers between the two specified numbers.
- A list of numbers separated by commas specifies all of the numbers listed.
- An asterisk specifies all legal values.

In the command field (sixth field), a percent sign (%) is translated to a new-line character. Only the first line of a command field (character string up to the percent sign) is executed by the shell. Any other lines are made available to the command as standard input.

Figure A-9 shows a typical **/usr/spool/cron/crontabs/***logname* file. The data shown are the **root** file. The file entries support the **calendar** reminder service and basic networking. Remember, you can use the **cron** function to decrease the number of data terminal driven system administration tasks: include recurring and habitual tasks in your crontab file.

```
0 1 * * * /usr/bin/calendar -
41,11 * * * * /usr/lib/uucp/uudemon.hour > /dev/null
45 23 * * * ulimit 5000; /bin/su uucp -c "/usr/lib/uucp/uudemon.cleanup"
  > /dev/null 2>&1
1,30 * * * * /usr/lib/uucp/uudemon.poll > /dev/null
```

Figure A-9: Typical **/usr/spool/cron/crontabs/root** File

(Refer to the **crontab**(1) manual page in the *IRIS-4D User's Reference Manual* for additional information.)